3/22/94

E-9654

NASA Contractor Report 195291

Computer Code for Controller Partitioning With IFPC Application

A User's Manual

Phillip H. Schmidt and Asim Yarkhan
University of Akron
Akron, Ohio

March 1994

Prepared for
Lewis Research Center
Under Grant NAG3–11467 Errata

National Aeronautics and
Space Administration

E-8654
4/7/94

ERRATA

NASA Contractor Report 195291

Phillip H. Schmidt and Asim Yarkhan
University of Akron
Akron, Ohio 44325

The grant number for the aforesaid Contractor Report is corrected to Grant NAG3–1146.

ABSTRACT

This is a user's manual for the computer code for partitioning a centralized controller into decentralized subcontrollers with applicability to Integrated Flight/ Propulsion Control (IFPC). Partitioning of a centralized controller into two subcontrollers is described and the algorithm on which the code is based is discussed. The algorithm uses parameter optimization of a cost function which is described here. The major data structures and functions are described. Specific instructions are given. The user is led through an example of an IFPC application.

User's Manual — Table of Contents

Acknowlegment

The authors wish to express their appreciation to Dr. Sanjay Garg of NASA Lewis Research Center for his excellent technical guidance and support during the project described here.

1. Introduction

Large interconnected systems such as the flight/propulsion systems of modern aircraft often exhibit significant coupling between the various subsystems. One example of such a system is the Short Take-Off and Landing (STOL) aircraft wherein the forces and moments generated by the propulsion system provide control and maneuvering capabilities for the aircraft at low speeds. This strong coupling suggests that a centralized control design be used, however, a centralized controller which is designed for the integrated plant considering all the interconnections between the flight and propulsion subsystems may be of high order and may be difficult to implement and validate. Specifically, in aircraft design it is the responsibility of the engine manufacturer to ensure that the propulsion system will provide the desired performance when installed in the aircraft. The engine manufacturer thus needs a separate engine controller to be able to perform extensive testing to assure adequate performance and integrity in the presence of operational and safety limits. This requirement suggests the need for decentralized implementation of Integrated Flight/Propulsion Control (IFPC) systems.

One approach to integrated control design which combines aspects of centralized and decentralized control design approaches is currently being developed at the NASA Lewis Research Center [**1**]. This approach consists of first designing a centralized controller, so that all subsystem interconnections are accounted for in the initial design stage, and then *partitioning the centralized controller* into separately implementable decentralized subcontrollers for individual subsystems. Here, *partitioning* means representing the high-order centralized controller with two or more lower order subcontrollers which have input/output intercoupling such that the overall control law obtained on assembling the subcontrollers closely approximates the input/output behavior of the centralized controller.

The computer code described in this user's manual is designed specifically for IFPC application and the notation and terminology used here reflects that application. The software described here uses a parameter optimization method to match the performance of a centralized controller with a partitioned controller consisting of two decentralized subcontrollers for the flight and propulsion systems. This matching will be subject to certain subsystem design requirements. The structure is shown in Fig. 1.1, with optional feedback paths indicated by dotted lines.

In the decentralized, hierarchical controller partitioning structure shown in Fig. 1.1,

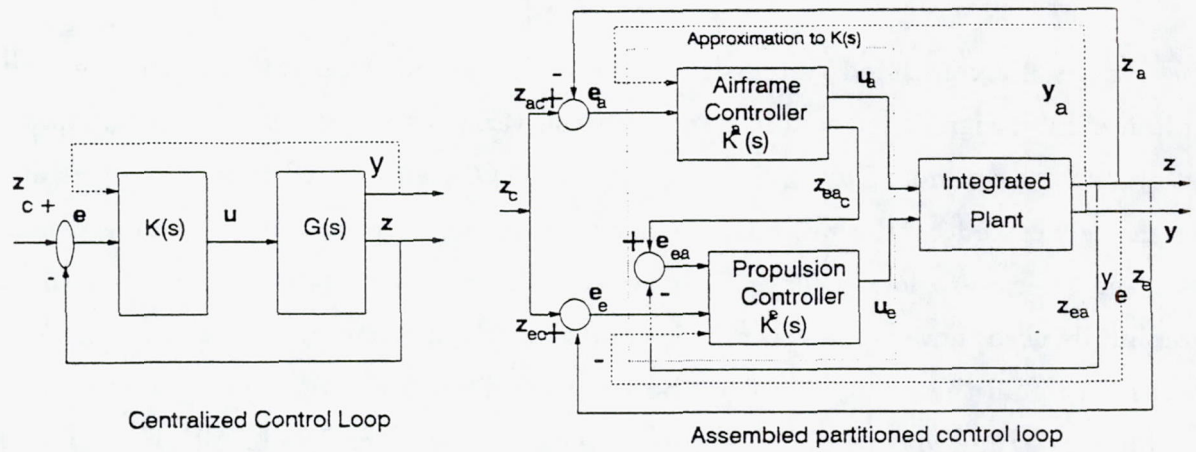


Figure 1.1 Controller Partitioning

the subscripts and superscripts "a" and "e" refer to airframe and propulsion (engine) quantities, respectively, and the subscript "c" refers to commanded quantities. The interface variables $\mathbf{z}_{ea}$ represent propulsion system quantities that affect the airframe, such as propulsive forces and moments. The structure is hierarchical in that the airframe (flight) controller produces commands for the engine controller via the interface variable ($\mathbf{z}_{ea_c}$) which are tracked by the propulsion subsystem.

Such a control structure allows the engine manufacturer to evaluate the engine subsystem performance independently of the airframe control and to verify that the engine subsystem will provide the desired performance when installed in the airframe. In general there are practical constraints on the achievable bandwidth of $\mathbf{z}_{ea}$ tracking for the engine subcontroller. A lower bound on the $\mathbf{z}_{ea}$ command tracking bandwidth is based on achieving the desired performance for the integrated system, while an upper bound is imposed by actuator limits and robustness requirements to high frequency modeling uncertainties.

The software discussed here refers to the structure described above. The variables are named according to the convention given above. The parameters in this optimization process are entries in the state-space representations of the subcontrollers. These parameters are bounded so as to maintain subcontroller (open-loop) stability. An assumption made in the formulation is that the plant has no direct feedthrough from control inputs, i.e. the plant "D" matrix is zero. This simplifies the determination of the cost function and its gradient.

One feature of the software is that the user may separately optimize the airframe

controller for a fixed engine controller or optimize the engine controller for a fixed airframe controller. The main alternative is to jointly optimize both although separate optimization is demonstrated in the example.

This user's manual is organized as follows. Section 2 briefly describes the cost function which is the objective to be minimized. The partitioning algorithm is described in section 3. There is also a brief description here of the interrelation among User-defined Functions (UDFs) so the user who wishes to change the cost function or partitioning structure will know which UDFs must be changed. Section 4 describes the data structures needed for carrying out partitioning using the MATRIX x programming language. This section also contains a description of the major data structures and variables used. Section 5 has a brief outline of the procedure for using the software. Section 6 contains a detailed example which exercises the algorithm showing applications of its options.

Appendix I contain a detailed discussion of the parameterization, the cost function and the gradient evaluation as they are implemented in the software. Appendix II contains short descriptions of the UDFs which implement the partitioning algorithm. Appendix III contains fully-documented source code for partitioning. Appendix IV contains the data file, INIT.DAT, and partitioned subcontrollers, [SKA_OPT, SKE_OPT] for the example in section 6.

2. The Cost Function to be Minimized

With reference to Figure 1.1, the partitioning problem can be stated as follows:

Given a *centralized controller* with transfer matrix $K(s)$ and a specification of the *partitioning structure* of controller inputs and outputs, i.e.

$$\begin{bmatrix} \mathbf{u}_a \\ \mathbf{u}_e \end{bmatrix} = K(s) \begin{bmatrix} \mathbf{e}_a \\ \mathbf{e}_e \\ \mathbf{y}_a \\ \mathbf{y}_e \end{bmatrix} \tag{2.1}$$

where $\mathbf{u}_a \in R^{k_a}$, $\mathbf{u}_e \in R^{k_e}$, $\mathbf{e}_a \in R^{m_a}$, $\mathbf{y}_a \in R^{l_a}$, $\mathbf{e}_e \in R^{m_e}$ and $\mathbf{y}_e \in R^{l_e}$; a choice of *interface variables* $\mathbf{z}_{ea} \in R^{p_{ea}}$; a *plant* with transfer matrix $\widehat{G(s)}$ of the form

$$\widehat{G(s)} = \begin{bmatrix} G(s) \\ G_{ea}(s) \end{bmatrix} \quad \text{with} \quad \begin{bmatrix} \mathbf{z}_a \\ \mathbf{y}_a \\ \mathbf{z}_e \\ \mathbf{y}_e \end{bmatrix} = G(s) \begin{bmatrix} \mathbf{u}_a \\ \mathbf{u}_e \end{bmatrix} \text{ and } \mathbf{z}_{ea} = G_{ea}(s) \begin{bmatrix} \mathbf{u}_a \\ \mathbf{u}_e \end{bmatrix}$$

find *subcontrollers* with stable transfer matrices $K^a(s)$ and $K^e(s)$, i.e.

$$\begin{bmatrix} \mathbf{u}_a \\ \mathbf{z}_{ea_c} \end{bmatrix} = K^a(s) \begin{bmatrix} \mathbf{e}_a \\ \mathbf{y}_a \end{bmatrix} \quad \text{and} \quad \mathbf{u}_e(s) = K^e(s) \begin{bmatrix} \mathbf{e}_{ea} \\ \mathbf{e}_e \\ \mathbf{y}_e \end{bmatrix};$$

where $\mathbf{e}_{ea} = \mathbf{z}_{ea_c} - \mathbf{z}_{ea}$, so that the closed-loop performance with the subcontrollers closely matches that with the centralized controller within the requirements of the subsystem.

The particular subsystem constraint for IFPC application is that the engine subcontroller $K^e(s)$ should have the structure of a command tracking controller for the interface variable commands $\mathbf{z}_{ea_c}$.

The cost function is formulated to reflect the difference between the centralized and partitioned controllers. The state space representations of the subcontrollers $K^a(s)$ and $K^e(s)$ are parameterized and the cost function is minimized over those parameters denoted as a vector $\mathbf{p}$. The formulation of this parameterization is discussed in Appendix I. Stability robustness may be achieved through the use of optional (user-provided) weighting matrices and a normalization function in determining the partitioning cost. Specific details

concerning the cost function, the parameters involved and the evaluation of the gradient of the cost are contained in Appendix I.

The cost function $f(\mathbf{p})$ is the sum of the performance cost, $f_0(\mathbf{p})$, and an additional cost of tracking the airframe-to-engine commands, $f_1(\mathbf{p})$, $f(\mathbf{p}) = f_0(\mathbf{p}) + f_1(\mathbf{p})$.

The performance cost, f_0, is the H_2 norm of the weighted (and possibly normalized) difference of transfer matrices for the centralized and partitioned controllers

$$f_0(\mathbf{p}) = \int_0^\infty \frac{1}{N_{\mathrm{PERF}}(\omega)} \mathrm{tr}\left[\left(W_o(j\omega)(K(j\omega) - \widetilde{K(\mathbf{p})}(j\omega))W_i(j\omega)\right)^*\right.$$
$$\left.\left(W_o(j\omega)(K(j\omega) - \widetilde{K(\mathbf{p})}(j\omega))W_i(j\omega)\right)\right] \mathrm{d}\omega \quad (2.2)$$

where K is the transfer matrix from the $\begin{pmatrix} \mathbf{e} \\ \mathbf{y} \end{pmatrix}$ inputs to the $\mathbf{u}$ outputs for the centralized controller, and $\widetilde{K}(s)$ is the transfer matrix of an "equivalent" centralized controller (having the same input/output structure as K) obtained by assembling the partitioned subcontrollers using appropriate plant information. Details of the state space representation for $\widetilde{K}(s)$ are given in Appendix I.

We are using the H_2 norm of the weighted difference between the transfer matrices for the centralized controller and the equivalent heirarchically partitioned subcontrollers as will be described in Appendix I. Since this difference must be strictly proper in order to apply this norm, it is reasonable for the D matrices for the centralized and partitioned controllers to be the same. Thus it may be desirable to fix the values of D^a_{aa}, $D^a_{ay_c}$, D^e_{ee}, and $D^e_{ey_c}$ (as described in Appendix I) to values determined directly by the centralized controller. This is one of the options available in "fixing the D-parameters".

$W_i(j\omega)$ and $W_o(j\omega)$ are optional input and output weighting matrices, $N_{\mathrm{PERF}}(\omega)$ is an optional scalar normalization function. For example, the weighting

$$W_i(s) = G(s)\left(I + K(s)G(s)\right)^{-1}$$

has been shown by Dale Enns [**2**] to lead to stability robustness for the partitioned system provided that the centralized system has this property. Other weighting and normalization will be discussed with the example.

$f_1(\mathbf{p})$ is the cost of tracking the $\mathbf{z}_{ea_c}$ command generated by the airframe subcontroller for the engine subcontroller. This cost minimizes the difference between the transfer matrices for the responses to the $\mathbf{z}_{a_c}$ command of $\mathbf{z}_{ea_c}$ using the partitioned controller and

$\mathbf{z}_{ea}$ using the centralized controller.

$$f_1(\mathbf{p}) = \int_0^\infty \sum_i \frac{\lambda_i}{N_{\text{TRACK}\,i}(\omega)} \left(||T^i_{\text{cent}}(j\omega) - \widehat{T^i}(\mathbf{p})(j\omega)||_2 \right)^2 \mathrm{d}\omega \tag{2.3}$$

T^i_{cent} is the transfer function vector from the airframe commands $\mathbf{z}_{a_c}$ to the i^{th} interface variable $\mathbf{z}_{ea}{}^i$ with the centralized controller. $\widehat{T^i}$ is the i^{th} row of the transfer function matrix $\widehat{T}$ from the airframe commands $\mathbf{z}_{a_c}$ to the interface variables as commanded by the partitioned airframe controller, $\mathbf{z}_{ea_c}$, with the partitioned subcontrollers. λ_i is a scalar weighting which determines the influence of f_1 on the total cost and $N_{\text{TRACK}\,i}(\omega)$ are (optional) scalar normalization functions. $||\cdot||_2$ denotes the Euclidean norm of the row vector. Here, one may use the normalizations $N_{\text{TRACK}\,i} = ||T^i_{\text{cent}}||_2^2$ to provide adequate scaling for this cost. The parameters λ_i provide weighting for the contribution of the tracking cost to the total cost. It was shown in [**3**] that manipulating λ_i provides an indirect means for maintaining reasonable bounds on the $\mathbf{z}_{ea_c}$ command tracking bandwidth.

It may be required that the engine subsystem be proper, a condition which would be violated if D^e_{eea} (described in Appendix I) is nonzero. As a result of the optimization process, D^e_{eea} may become large. This possibility is removed by "fixing $D^e_{eea} = 0$" when the option is presented while running the code.

3. The Partitioning Algorithm and its Implementation

The objective is to minimize the cost $f(\mathbf{p}) = f_0(\mathbf{p}) + f_1(\mathbf{p})$ as described above where the parameters $\mathbf{p}$ are certain entries in the state space representation matrices for $K^a(s)$ and $K^e(s)$ (denoted SKA and SKE in the code).

The fixed data used by the algorithm are state-space representations for the plant transfer matrix $G(s)$, the centralized controller $K(s)$, the (optional) weighting matrices $W_i(s)$ and $W_o(s)$ (denoted as SP, SC, SWI and SWO respectively in the code), as well as a partitioning structure for the numbers of controller inputs (airframe, MA; and engine, ME), numbers of outputs (airframe, KA; and engine, KE), numbers of plant measurements (airframe, LA; and engine, LE) and numbers of airframe to engine subcontrollers interface variables (PEA). The control designer may also introduce normalization functions (N_PERF and N_TRACK) for the performance and tracking costs. Examples of normalizations are given with the example in Section 6. The user must also enter values of the tracking weight parameters λ_i which determine the relative contribution of f_1 to the total cost.

The algorithm incorporates the Broyden-Fletcher-Shanno-Goldfarb (BFGS) quasi- Newton method to select directions of search for Fletcher's inaccurate linesearch, see [4]. This iterative method requires the calculation of the combined cost and its gradient for the parameters $\mathbf{p}$ as referred to above. It uses successive gradients to build up an approximation to the inverse Hessian matrix. Moreover, the inaccurate linesearch assures an adequate reduction in the cost function at each step without using excessive effort searching for a minimum far away from the ultimate solution. In this way, subsequent search steps are successively closer to those generated by Newton's Method and convergence is accelerated as the iterations proceed.

The flow of the parameter optimization algorithm for controller partitioning is shown in Figure 3.1. The main steps in the algorithm are:

1. The initial partitioning is obtained by applying the stepwise procedure described in [5]. Special attention is paid to obtaining reasonably low-order subcontrollers which are stable and satisfy the $\mathbf{z}_{ea}$ command-tracking requirement. The initial state-space representations for the transfer matrices $K^a(s)$ and $K^e(s)$ are denoted as S_KA and

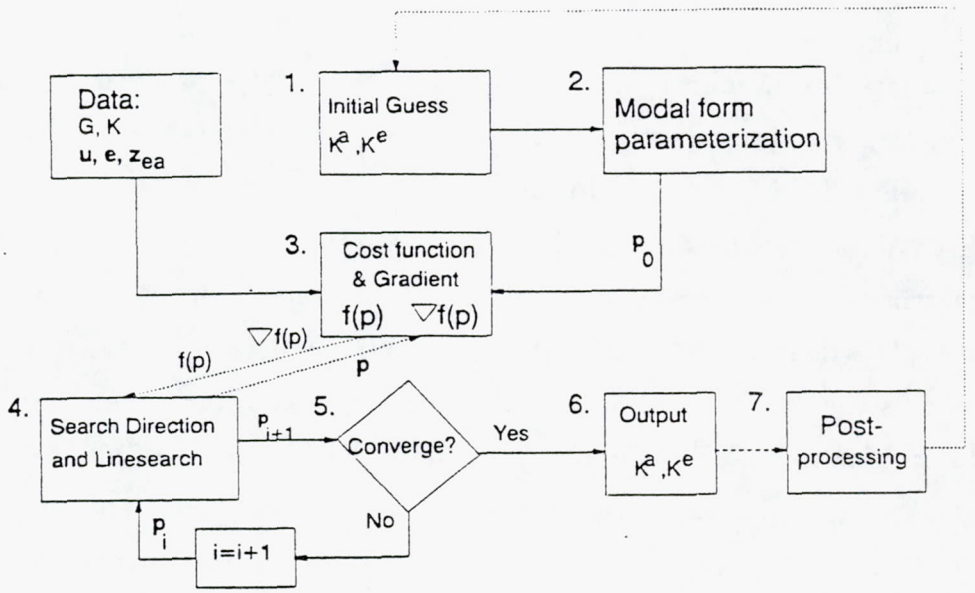


Figure 3.1 Flowchart for Partitioning Optimization Algorithm

S_KE; note that these are different from the working representations SKA and SKE described next.

2. The initial partitioning is converted to a "minimal parameter" form (with state-space representations SKA and SKE respectively) and used to generate an initial value of the parameter vector P_I. This form is described in Appendix I.

3. The initial (as well as any subsequent) value of the parameter vector is passed to a function which determines the state-space representation for the equivalent partitioned controller and calculates the combined cost, f (denoted FP in the code). The gradient (denoted as DFDP) is also computed analytically by the procedure described in Appendix I.

4. The BFGS method uses the current gradient in conjunction with previous information to generate a direction of search. The Fletcher inaccurate linesearch is carried out using the cost and gradient calculated at each parameter vector to predict a new parameter vector until one is found which yields a sufficient reduction in both the cost function and the size of the gradient. The new point is denoted as P_I1. This linesearch is constrained so as to maintain stability of the subcontrollers.

5. At the end of the linesearch, the new parameter vector and cost (P_I1 and FP_I1) are compared to the values at the beginning of the linesearch (P_I and FP_I) as a check on convergence. If the maximum change in all the parameters is less than a user-specified value and the change in the total cost is less than another value (MAX(P_I1-P_I) < EPSILON and ABS(FP_I1-FP_I) < DELTA) then convergence is declared and the iteration ceases. If in addition, the maximum absolute value of the partial derivatives is less than a user-specified tolerance (ABS(DFDP) < ETA), this is also noted. If the number of iterations exceeds ITER or the function value is sufficiently reduced (FP < FMIN), then the procedure stops with an appropriate message. If the convergence test fails, the algorithm proceeds to update the information used to determine the direction of search and to use the most recent cost and gradient values to generate a new direction of search and carry out the linesearch via steps 3. and 4.

6. The output of the algorithm is the state-space representation for subcontrollers (denoted SKA_OPT and SKE_OPT) which minimize the cost function $f(\mathbf{p})$ within the convergence criteria.

7. These subcontroller transfer matrices have the same orders n_a and n_e as the initial partitioning. Controller reduction can be performed on these "optimal subcontrollers" and the process of optimization can be repeated on the "new initial partitioning".

The algorithm is implemented in MATRIX$_X$ using a set of functions which are referred to in MATRIX$_X$ parlance as User-Defined Functions (UDFs). A glossary of variables and UDFs follows in Section 4. More complete descriptions of the UDFs appear in Appendix II and an annotated version of the code appears in Appendix III. The flow of the MATRIX$_X$ partitioning code as illustrated in Figure 3.2 follows:

a. The function START is called with input LAMBDA (required tracking cost weight scalar or vector) and STOP (optional stopping criteria vector).

   i. A file (INIT.DAT) is read to acquire the fixed data and initial partitioning (S_KA and S_KE) referred to above along with a three dimensional vector (FRQ) which gives the left and right end points as well as the number of logarithmically placed points in the interval over which numerical integration takes place.

   ii. START calls the routine MODL to put the initial partitioning into an appropriate

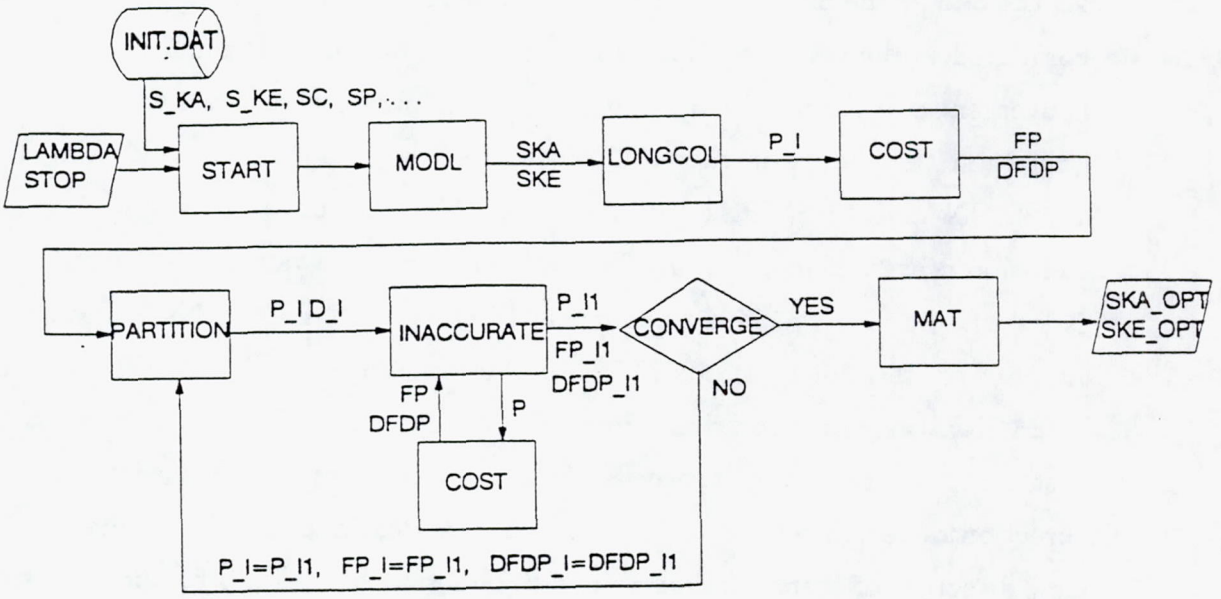


Figure 3.2 Flowchart for the MATRIX_X Partitioning Code

form to serve as a parameterization for the subcontrollers, i.e. SKA = MODL(S_KA) and SKE = MODL(S_KE).

iii. The function LONGCOL transforms these state-space representations for the subcontrollers into a parameter vector, i.e. P_I = LONGCOL(SKA, SKE). This function along with its "inverse" MAT ([SKA, SKE] = MAT(P_I)) are used throughout the code to transform between state-space representations for subcontrollers and a parameter vector. START also determines constants which will be used by the other functions and stores them in CONST.DAT .

iv. START then calls COST to calculate the initial cost (FP) and gradient (DFDP).

b. START calls PARTITIO with the convergence criteria (STOP) as input. PARTITIO is the main routine which

c. generates the search direction (DI),

d. calls the function INACCURATE which carries out Fletcher's inaccurate linesearch (bracketing/sectioning) using cost function, F(P_I+ALPHA*DI), and gradient values generated by the function COST. While INACCURATE is running, the user will see displayed first the bracketing interval (AL, ALPR) and function and derivative values (FAL, FPAL; FALPR, FPALPR) then the sectioning interval (A, B) and corresponding

function and derivative values (FA, FPA; FB, FPB). These give the user some sense of how rapidly the linesearch is progressing, but they can easily be removed from the code if desired.

e. Convergence/stopping conditions are checked by the function CONVERGE and steps c. — e. are repeated until they are satisfied. During these major iterations, the last twenty cost values are plotted.

f. The output of START is the final partitioning (SKA_OPT, SKE_OPT). At this point, the program ends.

The user may do a *posteriori* analysis such as order reduction on the subcontroller state-space representations (SKA_OPT, SKE_OPT) and start the procedure again with new data in the file INIT.DAT.

If the user wishes to modify the partitioning structure, changes will be necessary in the START, COST, LONGCOL and MAT routines. Different constants must be calculated and stored by START. Different formulations for the state space representation of $\widetilde{K}(s)$ and $\widehat{T}(s)$ must be coded in COST and new formulations for the gradient must be generated using the procedure described in Appendix I. Furthermore, the conversions between [SKA,SKE] and $\mathbf{p}$ by LONGCOL and MAT must be rewritten.
If a different formulation of the cost function is used, then only the portion of the COST UDF where the cost and gradient are computed must be changed. The algorithm requires a gradient with each evaluation of the cost. The new gradient may be the most difficult change to make.

4. Reference to Major Data Structures and Variables

Most of the input to the program is provided through the MATRIX x data file INIT.DAT. The following data structures are mandatory to the running of the program and must be provided in INIT.DAT prior to running the program.

INIT.DAT — Mandatory Data

SP, NP — the state space representation and the order for the integrated plant in system matrix form

$$\text{SP} = \begin{pmatrix} AP & BP \\ CP & DP \end{pmatrix}.$$

SC, NSC — the state space system matrix for the centralized controller, and its order.

S_KA, NS_KA — the state space system for an initial "guess" at the airframe controller, and its order.

S_KE, NS_KE — the state space system for an initial "guess" at the engine controller and its order.

PEA — the number of interface variables from flight controller to the engine controller.

FRQ — a vector of the form [FRQ(1);FRQ(2);FRQ(3)] where FRQ(3) logarithmically placed frequency points over the interval FRQ(1) $< \omega <$ FRQ(2) are used in the numerical integration for determining the costs. The number of points must be odd because of the numerical integration rule used.

The following data structures are optional and may be entered in INIT.DAT if desired. Indexing rules of MATRIX x do not allow the index zero or empty vectors. Thus, if some of the optional quantities are absent or have value zero, the code will place *dummy variables* in appropriate matrices and set corresponding size variables to nonzero quantities (usually one).

INIT.DAT — Optional Data — if absent, the indicated default values are set by the code.

SWI, NWI — state space system for input weighting of the difference between centralized and assembled partitioned controllers, and its order. *If absent, the code sets SWI = an identity matrix of size (MA + ME + LA + LE + 1) and NWI = 1.*

SWO, NWO — state space system for output weighting of the difference between centralized and assembled partitioned controllers, and its order. *If absent, the code sets SWO = an identity matrix of size (KA + KE + 1) and NWO = 1.*

LA — number of integrated plant measurements to the airframe controller. *If absent, the code sets LA = 1; appropriate zero entries are introduced in SP, SC, and SKA. This allows for the case where there are no measurements fed from the plant to the airframe controller.*

LE — number of integrated plant measurements to the engine controller. *If absent, the code sets LE = 1; appropriate zero entries are introduced in SP, SC, and SKE. This allows for the case where there are no measurements fed from the plant to the engine controller.*

NPERF — normalization vector of size (FRQ(3) × 1) for the performance cost. *If absent, the code sets NPERF to a vector of ones.*

NTRACK — normalization matrix of size (FRQ(3) × PEA) for the tracking cost. *If absent, the code sets NTRACK to a matrix of ones.*

STABIL — a necessarily negative parameter which is used to guarantee that all eigenvalues of the subcontrollers have negative real parts for stability. *If absent, the code sets STABIL* $= -10^{-9}$.

The major constants used within the code are created by the execution of START and stored in the file CONST.DAT.

MA, ME — number of airframe and engine controller inputs.

LA, LE — number of integrated plant measurements to airframe and engine controllers.

KA, KE — number of airframe and engine controller outputs.

PEA — number of intermediate commands from airframe controller to engine controller.

SP, NP — state space system for the integrated plant, with its order.

S_KA, NS_KA — state space system for the initial airframe controller (K^a) in modified

modal form (or in original form if the system is to be held fixed) along with its order.

S_KE, NS_KE — state space system for the initial engine controller (K^e) in modified modal form (or in original form if the system is to be held fixed) along with its order.

SK, NK — state space system for centralized controller (K), with its order.

SG, NG — state space system for transfer matrix from $\mathbf{z}_{a_c}$ to $\mathbf{z}_{ea}$ (T_{cent}) using the centralized controller, with its order.

WEIGHT — a vector containing weights to use with Simpson's integration rule; depends on FRQ(3) for the number of points at which integration is to take place.

OMEGA — the vector of logarithmically placed frequency points at which sampling is to be done; determined by FRQ.

AORE — flag to indicate whether the airframe controller (1) or the engine controller (2) or neither (0) is held fixed during the optimization process.

FIXD — flag to indicate whether the 'D' matrices are held fixed; if FIXD = 0 none are fixed, if FIXD = 1 or 2 the DAA, DAYA, DEE and DEYE are held fixed during the optimization process and (if FIXD = 1) then DEEA and DEAE are variables or (if FIXD = 2) then DEEA is set to a zero matrix and DEAE is a variable matrix; if FIXD = 3 then only DEEA is set to a zero matrix and all the remaining D's are variable matrices. In any case DAA, DAYA, DEE and DEYE are saved in CONST.DAT.

STABIL — a necessarily negative parameter which is used to guarantee that all eigenvalues of the subcontrollers have negative real parts for stability. *If absent, the code sets STABIL* $= -10^{-9}$.

The outputs of interest to the program are kept in the file INTER.DAT, which contains a history of the optimization process and information which can be used to restart the program (after a crash or after intentionally stopping it) if desired.

INTER.DAT contains

P_I — the last point (parameter vector) that met the inaccurate linesearch minimization criteria, that is, the i^{th} point. P_I is a column vector consisting of the successive α, β values in the 2×2 blocks of A^a and A^e, the successive columns of B^a_{aa} after the first, the successive columns of C^a, the successive columns of D^a corresponding to the $\mathbf{z}_{ea_c}$ outputs, the successive columns of B^e after the first, the successive columns of C^e_{ee} and the successive columns of D^e corresponding to the $\mathbf{e}_{ea}$ inputs in this order. See Appendix I for a more complete discussion of the parameter vector as it is related to the cost function.

JH0 — the complete history of the cost of partitioning through the i^{th} iteration.

JH1 — the complete history of the cost of tracking through the i^{th} iteration.

FH — the complete history of the total cost, FP = FP0 + FP1 through the i^{th} iteration.

GRAD0 — the gradient of the partitioning part of the cost function, f_0, at the i^{th} iteration.

GRAD1 — the gradient of the tracking part of the cost function, f_0, at the i^{th} iteration.

GRADI — the total gradient of the cost function at the i^{th} iteration.

LAMBDA — the PEA$\times$1 vector which weights the contribution of the airframe to engine command tracking cost in the total cost function.

HI — the inverse Hessian matrix being used during the BFGS optimization process.

FX — keeps track of which parameters corresponding to A^a and A^e are at the stability bound.

Refer to the cost section of Appendix I for the structures of SP, SK, SKA, SKE and SG.

5. Instructions to the User

The following instructions are intended to be a quick introduction to the code. They do not attempt to explain the details of what is occurring during the execution. More thorough documentation of the MATRIX$_X$ functions is available in the documented code contained in Appendix III.

- Use MATRIX$_X$ to construct the data file INIT.DAT described in the preceding section. The initial approximation to the subcontroller state-space matrices S_KA and S_KE can be produced by the procedure described in [**5**] or may come from a previous application of the partitioning software. If any of the optional data are not present in INIT.DAT then the code will produce the defaults indicated earlier.

- Start MATRIX$_X$ and type

  DEFINE 'START.MTX'

  to activate the code. To execute START you must enter a value of LAMBDA, a weighting of the tracking cost relative to the total cost. LAMBDA can be a PEA $\times$ 1 vector whose entries individually weight the $\mathbf{z}_{ea}$ output responses to the total $\mathbf{z}_{a_c}$ inputs. If all the weights are to be the same then a scalar may be entered. This variable emphasizes the degree to which the tracking cost will affect the total cost.

  Optionally, the vector STOP of stopping conditions can be defined.

  STOP = [EPSL; DELTA; ETA; ITER; FMIN].

  The program stops if the following criteria are met:

  the maximum change in the parameters MAX(|P_I-P_I1|) < EPSL and

  the change in the cost |FP_I-FP_I1| < DELTA and

  the norm of the gradient |DFDP_I| < ETA or

  the number of major iterations I > ITER or

  the cost FP_I1 < FMIN

  If STOP is not entered, the following values are set by the code

  STOP = [EPSL; DELTA; ETA; ITER; FMIN] = [10^{-9}; 10^{-9}; 10^{-9}; 100; 0.1].

- Now the program can be executed by entering

  [SKA_OPT,SKE_OPT]=START(LAMBDA, STOP) or

  [SKA_OPT,SKE_OPT]=START(LAMBDA).

  The choice of fixing the engine or airframe system matrices will be presented. If neither is to be fixed (the usual choice) enter 0.

  Various options for fixing the $D-$submatrices are presented. If none are to be fixed enter 0 here.

  During the linesearch procedure, the user will see function and derivative values which indicate the progress of the search for a reduction in the function value.

  After every major iteration of the program (starting from the second), a MATRIX x graph is generated showing the costs (total, partitioning and tracking) for the last twenty iterations.

- When the run ends, the output of START is the final optimized state-space representations for the subcontrollers SKA_OPT and SKE_OPT.

  The data file INTER.DAT stores the history of the costs, the most recent parameter vector, approximation to the inverse Hessian matrix and gradient. This information can be used to see the progress of the algorithm, and to restart the program either after a successful termination or after user interruption.

There is a routine called RESTART which is available for restarting from a system crash or an intentional interruption. It uses the data stored in INTER.DAT and CONST.DAT and allows the user to define new values of the tracking cost weight λ, the stopping criteria STOP, and/or a new inverse Hessian approximation.

- The inputs and outputs of the RESTART routine are similar to those for START. There are three alternatives:

  1) enter [SKA_OPT,SKE_OPT]=RESTART(LAMBDANEW) if all that is changed is the LAMBDA weighting parameter or

  2) enter [SKA_OPT,SKE_OPT]=RESTART(LAMBDANEW, STOP) if a change is made in LAMBDA and/or STOP or

  3) or enter [SKA_OPT,SKE_OPT]=RESTART(LAMBDANEW, STOP, 1) if a restart with the identity matrix as the initial approximation to the inverse Hessian is desired. Note that you must enter values of LAMBDANEW and STOP even if they are the same as the previous LAMBDA and STOP.

The restarted program executes in the same way as before, with results stored in the file INTER.DAT and the output of the program being the optimized state-space representations SKA_OPT and SKE_OPT.

6. Example of Controller Partitioning

STOL Example.

The controller partitioning software is first applied to a centralized flight/propulsion controller for a STOL aircraft as was described in reference [**6**]. This controller has the form $\mathbf{u} = K(s)\mathbf{e}$ with the error vector $\mathbf{e}$ consisting of errors, $\mathbf{e} = [e_v, e_q, e_{N2}, e_{EPR}]^\top$, in following velocity (v), pitch rate variable ($q_v = q + 0.1\theta$), engine fan speed ($N2$) and engine pressure ratio (EPR) commands. The control input vector $\mathbf{u}$ consists of rates of change of thrust vectoring angle, fuel flow, thrust reverser port area and nozzle throat area, $\mathbf{u} = [\dot{\delta_{TV}}, \dot{WF}, \dot{A78}, \dot{A8}]^\top$. $\mathbf{u}$ consists of rates because integrators were appended to the control inputs during the process of centralized control design to achieve zero steady-state error for step commands. The partitioned airframe and engine controllers are desired to have inputs $\mathbf{e}_a = [e_v, e_q]^\top$ and $\mathbf{e}_e = [e_{N2}, e_{EPR}]^\top$ and outputs $\mathbf{u}_a = [\dot{\delta_{TV}}]$ and $\mathbf{u}_e = [\ \dot{WF}, \dot{A78}, \dot{A8}]^\top$ respectively. The interface variable $\mathbf{z}_{ea}$ for this example is the single variable FEX, the axial thrust generated by the propulsion system. An initial controller partitioning was obtained using the procedure discussed in [**5**].

The numbers of airframe and propulsion subcontroller inputs are thus (MA=)2 and (ME=)2 while the subcontrollers have (KA=)1 and (KE=)3 outputs respectively. There is (PEA=)1 interface variable and no direct measurements are fed back from the integrated plant to the subcontrollers (LA and LE are absent from INIT.DAT since there are no measurements).

State-space matrices for the integrated plant, SP of order (NP=)13, the centralized controller, SC of order (NSC=)13 and initial partitioning, S_KA of order (NS_KA=)10 and S_KE of order (NS_KE=)7 are listed in Appendix IV.

The optimization is done over the frequency range $\omega \in [0.1, 100]$ with 41 frequency points (FRQ=[0.1; 100; 41]). The frequency weighting

$$W_i(s) = G(s)(I + K(s)G(s))^{-1}$$

is used to achieve good performance matching as well as stability robustness for the equivalent controller. The state-space representation for this weighting, SWI of order (NWI=)26, is obtained from SP and SC. The tracking normalization, N_TRACK= $||T_{\text{Cent}}(j\omega)||_2^2$, is used to scale the tracking cost. The state-space representation for $T_{\text{Cent}}(s)$ is constructed as in expression (I.3) in Appendix I. Since LA and LE are absent, any blocks involving $\mathbf{y}_a$ or $\mathbf{y}_e$ as either inputs or outputs are omitted. Notice that since PEA=1, T_{Cent} is a $1\times$ MA row vector. Neither output weighting (SWO) nor performance normalization (N_PERF) were used.

All the necessary variables are stored in INIT.DAT .

The tracking weighting parameter LAMBDA is set to 0.05 and the stopping criteria vector is defined as

STOP=[1e-9;1e-9;1e-9;100;0.1].

After defining the UDF START by entering

define 'START.MTX'

the program is started by entering

[SKA,SKE]=START(LAMBDA,STOP).

Respond to the question concerning fixing the engine or airframe by entering '0' to fix neither as shown in Figure 6.1.

The '0' response to the question concerning the $D-$ matrices as shown in Figure 6.1 will fix none of these submatrices . The program now begins.

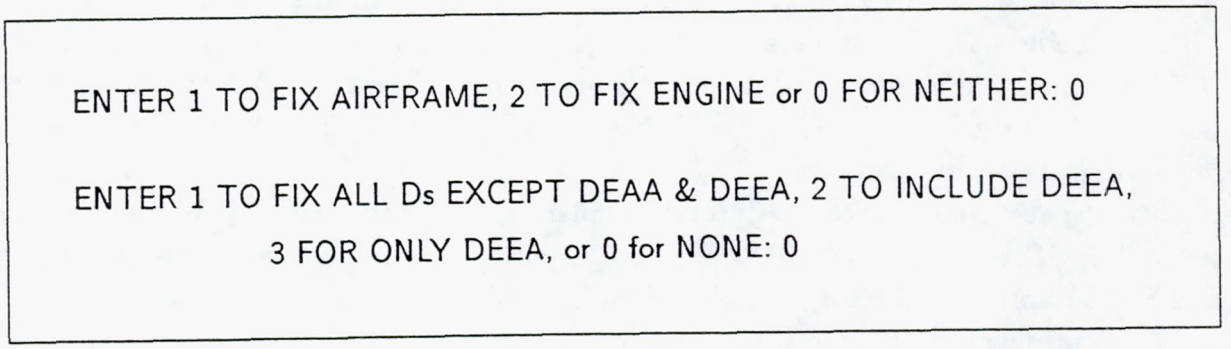


Figure 6.1 Screen After Responding to Questions

During a major iteration, the user will see values of AL, FAL, FPAL, ALPR, FALPR, A, FA, FPA, B, FB and FPB displayed on the screen. These values result from calculations during the linesearch as described in Step 4 of Section 3. In particular [AL, ALPR] is the interval used in the "bracketing" phase of the linesearch and [A, B] is the interval used during "sectioning". The function values (and directional derivatives) at the endpoints of these intervals are denoted by FAL, FALPR, FA, and FP (respectively FPAL, FPALPR, FPA and FPB). The user can follow the progress of the linesearch by viewing the values displayed on the screen as in Figure 6.2.

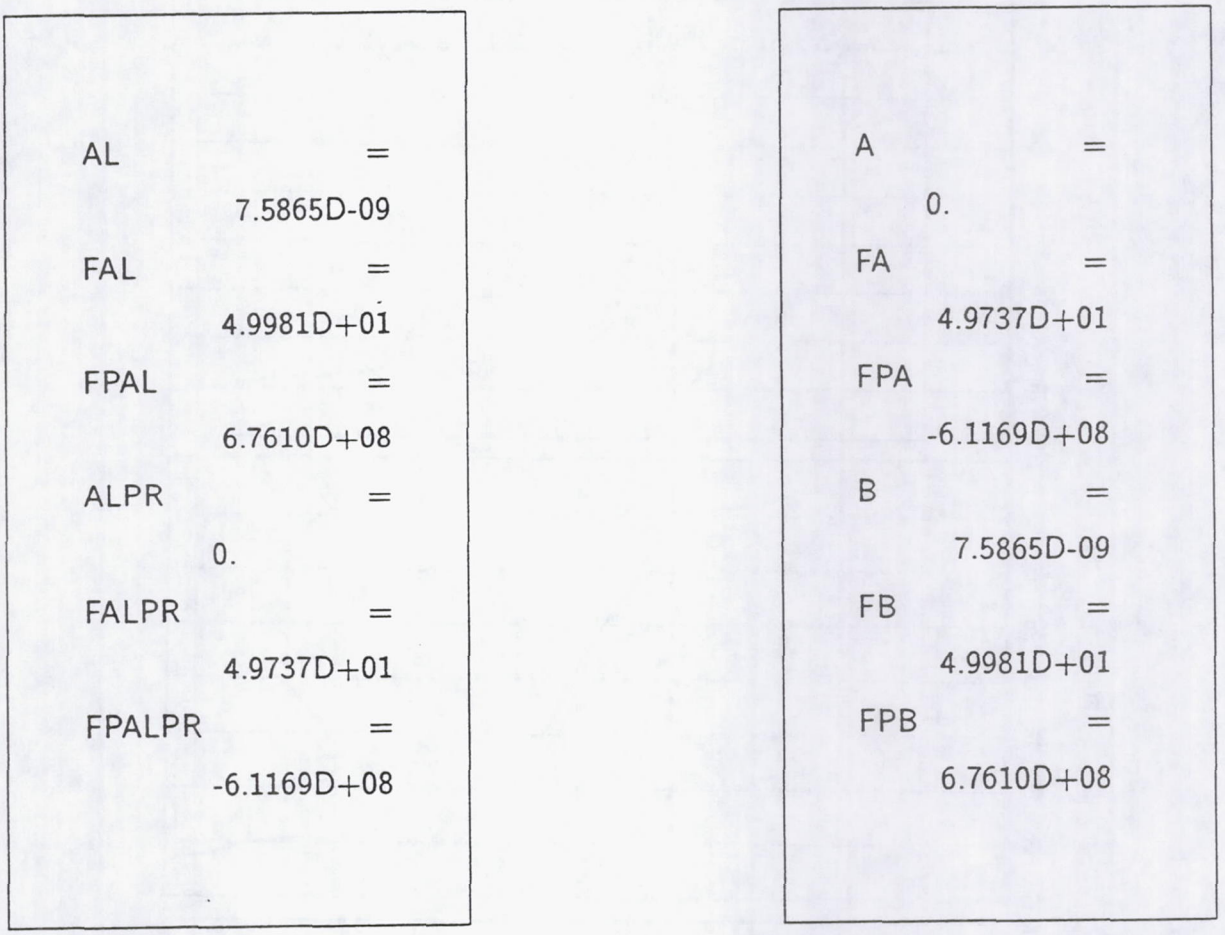


Figure 6.2 Screen Display During the Linesearch

After the first linesearch succeeds the user will see a graphical display of the values of the total, performance and tracking costs for the previous major iterations (after the twentieth, only the last twenty are displayed). A typical screen is displayed in Figure 6.3.

After 100 iterations, SKA and SKE are returned. The convergence criteria were not all met, rather the program stopped because the maximum number of iterations was reached. Nonetheless, as will be seen by a *posterioiri* analysis, the resulting subcontrollers exhibited good performance and tracking properties. The total cost history is shown in Figure 6.4.

The performance of the initial controller partitioning is evaluated in comparison with that of the centralized controller by comparing closed-loop system response to step commands in the controlled variables **z**.

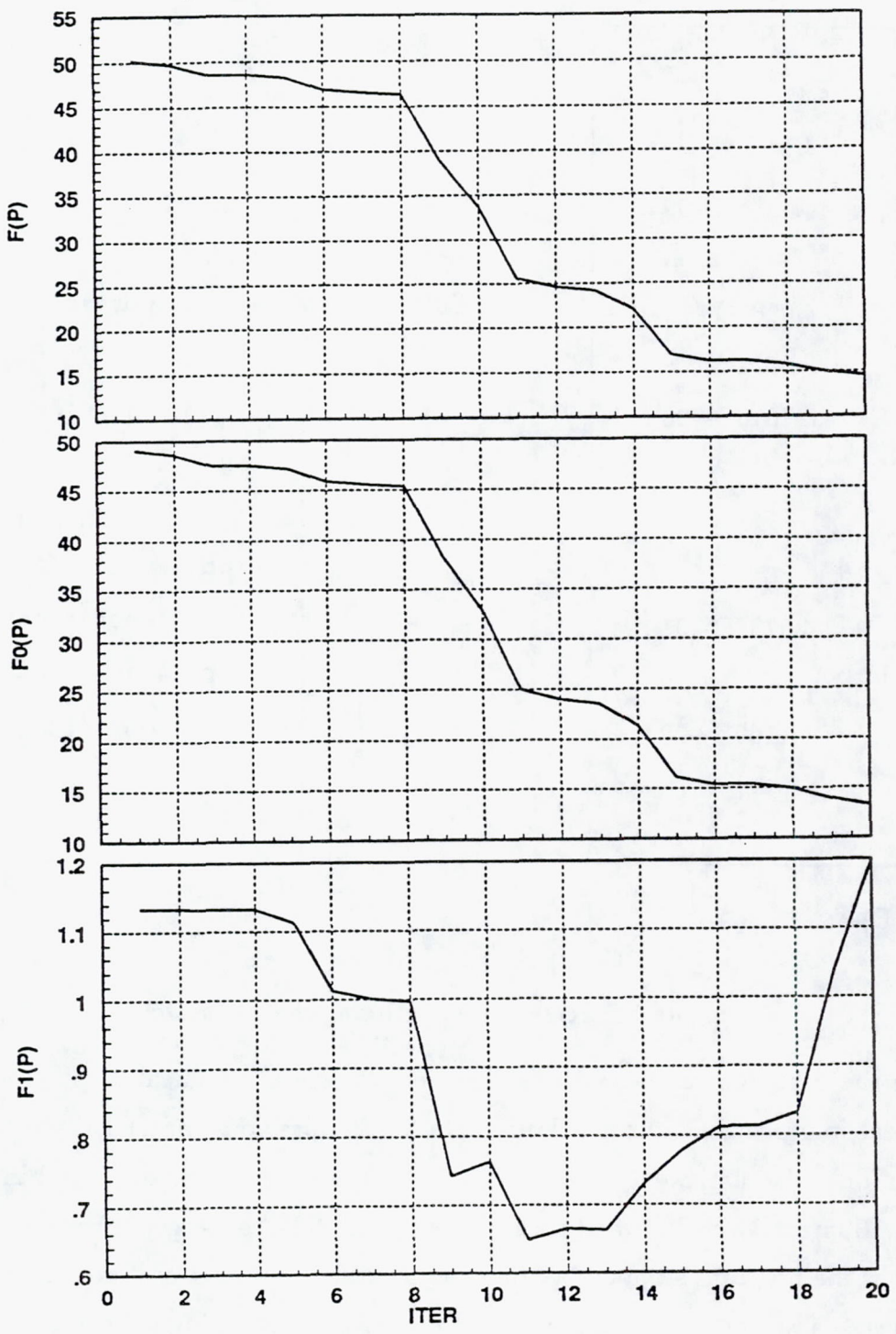


Figure 6.3 Screen Display Showing Total, Performance and Tracking Costs

The responses to q_{v_c}, $N2_c$ and EPR_c with the initial as well as the optimized parti-

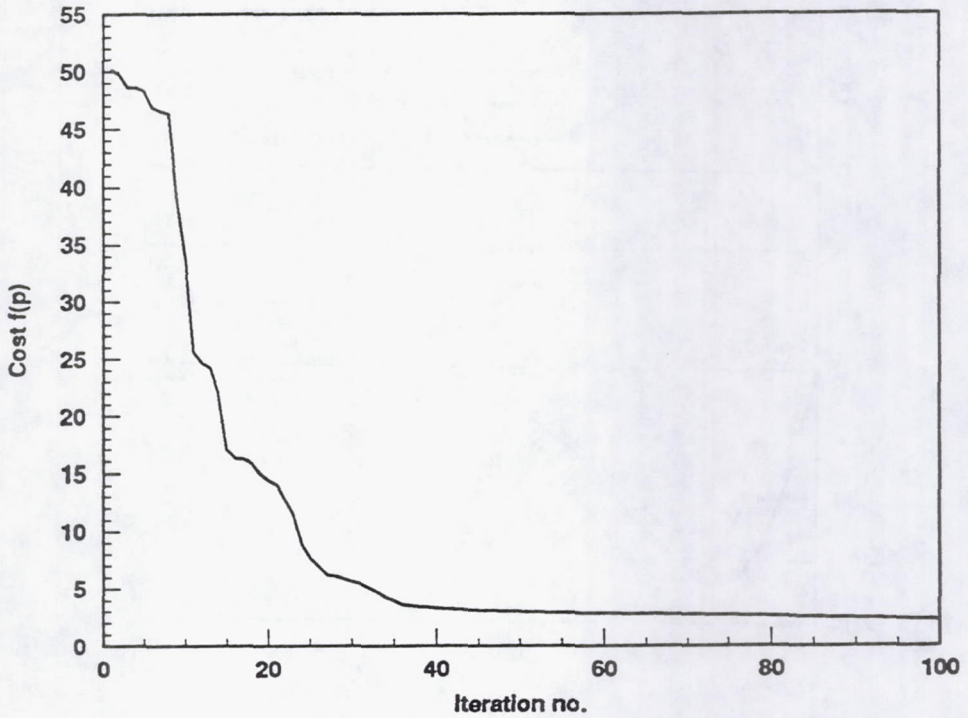


Figure 6.4 Cost History for Controller Partitioning Optimization

tioned subcontrollers were comparable to those with the centralized controller so they are not shown here. However, the responses to V_c, shown in Fig. 6.5, show considerable degradation in terms of increased coupling in the $N2$ and EPR responses with the initial partitioned subcontrollers. This deficiency was overcome by the optimized partitioned controllers as can be seen in Fig. 6.5. Note that all the quantities shown in Fig. 6.5 are normalized, using scalings discussed in [**6**], to allow a direct comparison of the various response magnitudes. In addition, the response of FEX (the interface variable) to V_c using partitioned subcontrollers was also comparable to that using the centralized controller as is seen in Fig. 6.6.

Since the performance with the optimized subcontrollers is found to be acceptable, an effort is made to reduce the orders of the subcontrollers. The engine subcontroller is reduced to 4^{th} order by residualization of the three high frequency modes without any loss of performance. Through the use of internally balanced reduction techniques [**7**], the airframe subcontroller is reduced to 6^{th} order (from the original 10^{th} order) without excessive mismatch in the controller transfer matrix characteristics as is seen from the

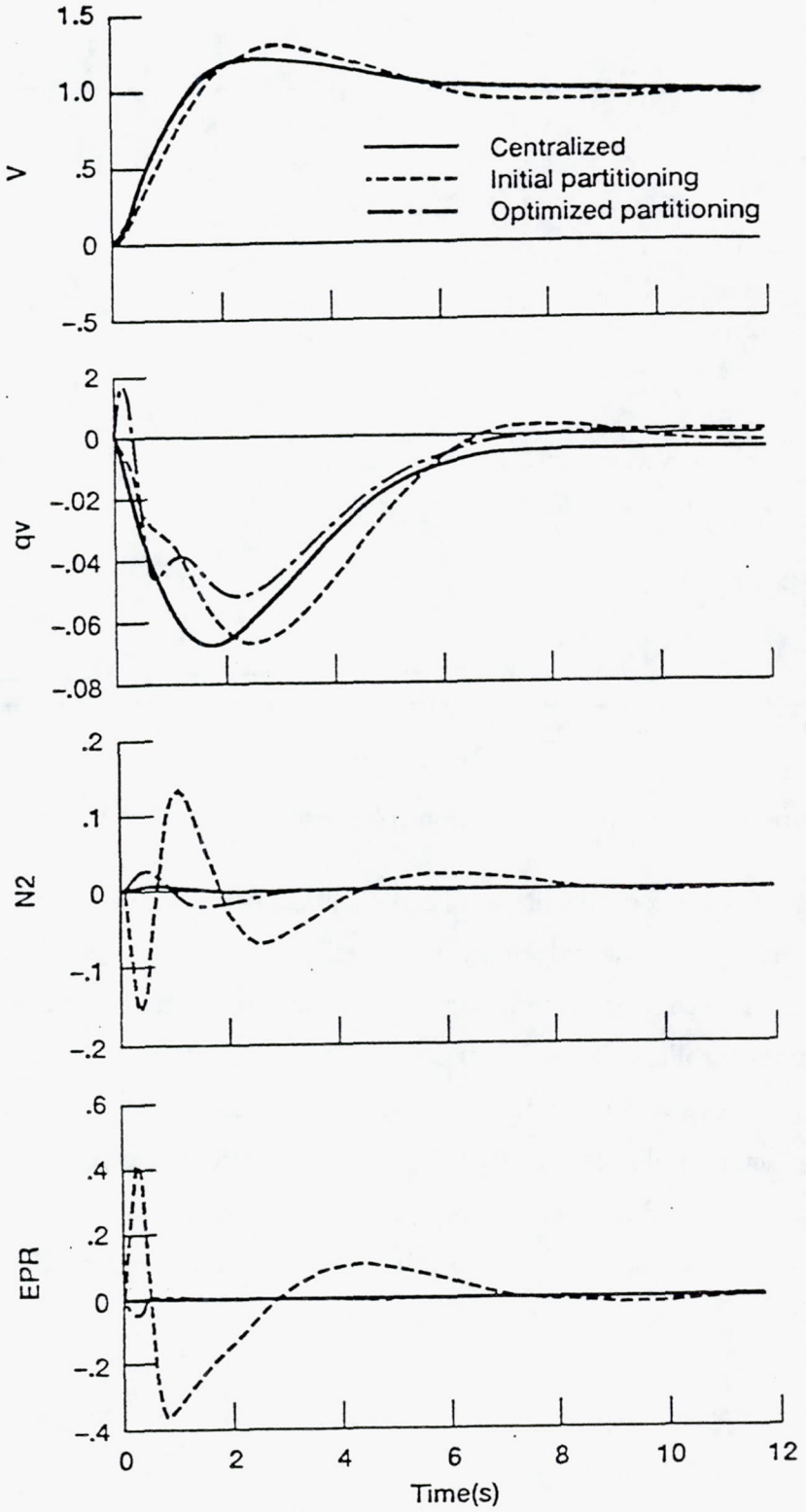


Figure 6.5 Closed-Loop System Response to Step Velocity Command for Centralized Controller, Initial Partitioning and Optimized Partitioning

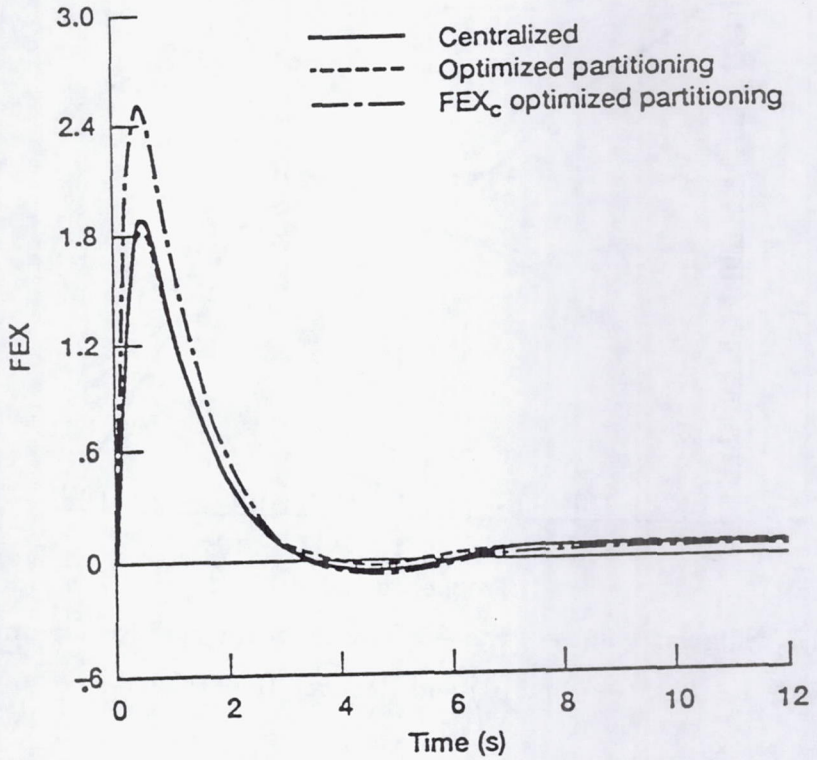


Figure 6.6 FEX Response to Step Velocity Command for Centralized Controller and Optimized Partitioning

full and reduced order airframe controller singular values comparison in Fig. 6.7. This reduced order airframe subcontroller does, however, exhibit deterioration in closed-loop performance in the V and q_v response comparison plots for a step change in V_c as shown in Fig. 6.8.

The reduced order subcontroller state-space matrices are stored as S_KA (order NS_KA=6) and S_KE (order NS_KE=4) in INIT.DAT and the program is started by entering

[SKA,SKE]=START(LAMBDA,STOP).

One should not use RESTART here since the CONST.DAT file will not contain data related to the reduced order subcontrollers. Moreover, since the engine subcontroller is acceptable, the optimization should take place over only the airframe subcontroller parameters. When requested to enter a value to fix a subcontroller, enter '2' to fix the engine subcontroller.

The program will execute as before, generating optimal SKA and SKE (fixed to the initial reduced order S_KE). The response obtained with the optimized reduced order airframe subcontroller for step V_c is shown in Fig. 6.8. Note that the optimized subcontroller

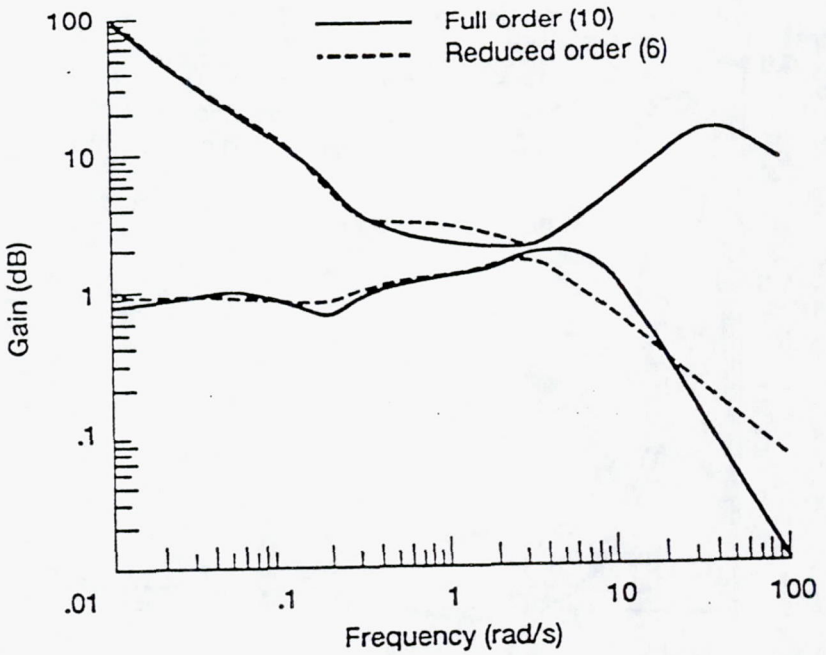


Figure 6.7 Singular Values of the Airframe Subcontroller for Optimized Partitioning with $\lambda = 0.05$ — Full (10) and Reduced (6) Orders

also provides improved tracking of the velocity command. The state-space matrix for the optimized reduced order airframe subcontroller is listed in Appendix IV.

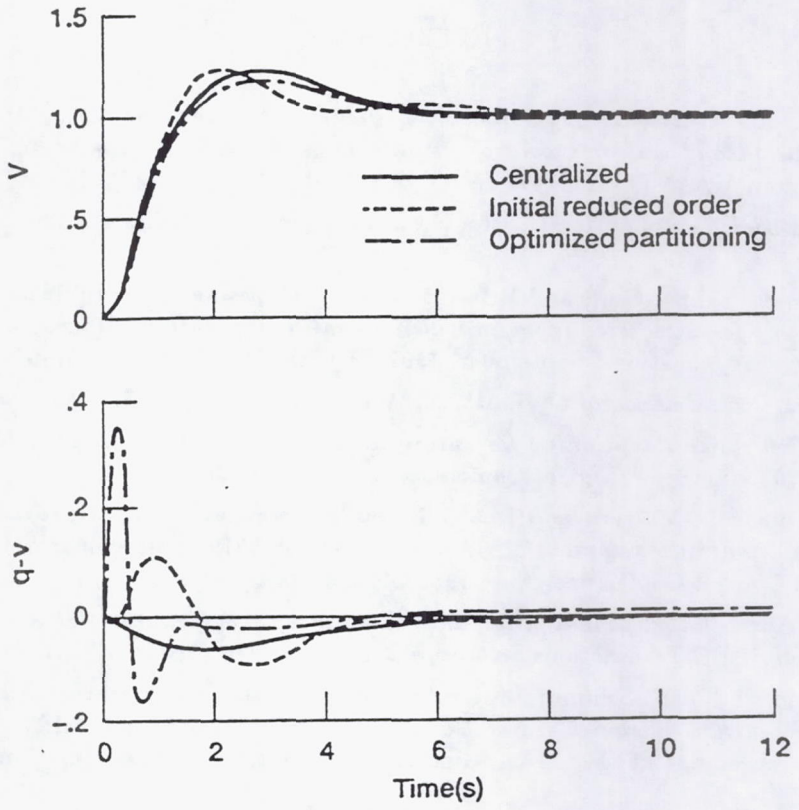


Figure 6.8 Closed-Loop System Response to Step Velocity Command for Centralized Controller and Reduced Order Partition Subcontrollers — Initial and Optimized

For a discussion of the application of this code to the design of a decentralized controller for a Short TakeOff and Vertical Landing (STOVL) aircraft, the user is referred to [8]. This application uses measurements from the integrated plant to the subcontrollers and includes more than one interface variable. In addition, several different weighting matrices are discussed in [8].

REFERENCES

[1] Sanjay Garg, Peter J. Ouzts, Carl F. Lorenzo and Duane L. Mattern, *IMPAC – An integrated methodology for propulsion and airframe control* , Proceedings of the 1991 American Control Conference, June, 1991, Boston, MA, **1** (1991), 747- 754.

[2] Dale F. Enns, *Model Reduction for Control Systems Design*, Ph.D. dissertation, Stanford University, 1984.

[3] Phillip H. Schmidt, Sanjay Garg and Brian Holowecky, *A parameter optimization approach to controller partitioning for integrated flight/propulsion control application*, Proceedings of the 1992 Conference on Control Applications, September, 1992, Dayton, OH, **2** (1992), 972-979.

[4] R. Fletcher, *Practical Methods of Optimization*, Wiley, New York, 1987.

[5] Sanjay Garg, *Controller Partitioning for Integrated Flight/Propulsion Control Implementation*, presented at the 1992 American Control Conference, June, 1992, Chicago, IL.

[6] Sanjay Garg, Duane L. Mattern and Randy E. Bullard, *Integrated flight/propulsion control system design based on a centralized approach*, AIAA Paper No. 89-3519, Proceedings of the AIAA Guidance, Navigation and Control Conference, Aug. 1989, Boston, MA.

[7] B.C. Moore, *Principal Component Analysis in Linear Systems: Controllability, Observability, and Model Reduction*, IEEE Transactions on Automatic Control **26** (1981), 17-31.

[8] Sanjay Garg and Phillip H. Schmidt, *Application of Controller Partitioning Optimization Procedure to Integrated Flight/Propulsion Control Design for a STOVL Aircraft*, AIAA Paper No. 93-3766, to appear in Proceedings of the AIAA Guidance, Navigation and Control Conference, Aug. 1993, Monterey, CA.

[9] Uy-Loi Ly, Arthur E. Bryson and Robert H. Cannon, *Design of low order compensators using parametric optimization*, Automatica **21** (1985), 315-318.

[10] Phillip H. Schmidt and Sanjay Garg, *Decentralized Hierarchical Partitioning of Centralized Integrated Controllers*, Proceedings of the 1991 American Control Conference, Boston, MA, June, 1991 **1** (1991), 755-760.

APPENDIX I

DEVELOPMENT OF THE COST AND ITS GRADIENT

The Parameters. Parameters in the optimization process are certain entries in state-space realizations of $K^a(s)$ and $K^e(s)$ as defined in the formula (I.1). The notation $\mathcal{M}^s_{\mathbf{oi}}$ is used throughout to indicate the matrix $\mathcal{M} \in \{A, B, C, D\}$ in the state-space realization of the system transfer matrix $s \in \{c, p, a, e\}$ (c =centralized controller, p =plant, a =airframe subcontroller, e =engine subcontroller) with input $\mathbf{i}$ (respectively output $\mathbf{o}$) $\in \{p, a, e, ea_c, ea\}$ (ea_c =interface variable commands, ea =interface variables).

$$K^a(s): \quad \begin{bmatrix} \mathbf{u}_a \\ \mathbf{z}_{ea_c} \end{bmatrix} = \left(\begin{pmatrix} C^a_{aa} \\ C^a_{ea_c a} \end{pmatrix} (sI - A^a)^{-1} \begin{pmatrix} B^a_{aa} & B^a_{ay_a} \end{pmatrix} + \begin{pmatrix} D^a_{aa} & D^a_{ay_a} \\ D^a_{ea_c a} & D^a_{ea_c y_a} \end{pmatrix} \right) \begin{bmatrix} \mathbf{e}_a \\ \mathbf{y}_a \end{bmatrix}$$

$$K^e(s): \quad \mathbf{u}_e = \left(C^e_{ee}(sI - A^e)^{-1}(B^e_{eea} \; B^e_{ee} \; B^e_{ey_e}) + (D^e_{eea} \; D^e_{ee} \; D^e_{ey_e})\right) \begin{bmatrix} \mathbf{e}_{ea} \\ \mathbf{e}_e \\ \mathbf{y}_e \end{bmatrix}.$$

For the purposes of this software description, the corresponding state space matrices are written as

$$\text{SKA} = \begin{bmatrix} A^a & B^a_{aa} & B^a_{ay_a} \\ C^a_{aa} & D^a_{aa} & D^a_{ay_a} \\ C^a_{ea_c a} & D^a_{ea_c a} & D^a_{ea_c y_a} \end{bmatrix} \quad \text{and} \quad \text{SKE} = \begin{bmatrix} A^e & B^e_{eea} & B^e_{ee} & B^e_{ey_e} \\ C^e_{ee} & D^e_{eea} & D^e_{ee} & D^e_{ey_e} \end{bmatrix} \tag{I.1}$$

One consideration in choosing a parameterization is to introduce a "minimal" number of parameters in the optimization process. A real canonical form used in [**9**] served as the model for our parameterization. The subcontroller system dynamics matrices A^a and A^e are represented as block diagonal matrices with two-by-two real companion blocks of the form $\begin{bmatrix} 0 & 1 \\ \alpha & \beta \end{bmatrix}$. If the order of either A^a or A^e is odd, there is also one diagonal real entry corresponding to a real eigenvalue. In addition, α and β are constrained to be negative in order to meet the requirement that subcontrollers be stable. It should be noted that this form for the A matrices does not allow for a Jordan block structure. However, since the matrices are obtained from a numerical process, it is improbable that the "optimal" solution would need such a special structure.

In addition the first columns of each of the subcontroller input matrices B^a_{aa} and B^e_{eea} are fixed at non-zero values determined as follows. State space representations for the initial partitioning are not required to be in the canonical form described above. For each of them, a similarity

transformation, T, is applied to the initial A matrix so that TAT^{-1} is in the proper form. If A is an $n \times n$ matrix then there will be n degrees of freedom in the determination of T. Different T will yield the same canonical TAT^{-1} but different transformed TB and CT^{-1} matrices. This implies that there are actually n degrees of freedom in the determination of TB and CT^{-1}. We select a simple T which is nonsingular and compute TB and CT^{-1}. We remove the degrees of freedom by fixing the n entries in the first column of TB to their values or 10^{-9} if the corresponding value is zero.

We are using the H_2 norm of the weighted difference between the transfer matrices for the centralized controller and the equivalent heirarchically partitioned subcontrollers as will be described below. Since this difference must be strictly proper in order to apply this norm, it is reasonable for the D matrices for the centralized and partitioned controllers to be the same. Thus it may be desirable to fix the values of D^a_{aa}, $D^a_{ay_a}$, D^e_{ee}, and $D^e_{ey_e}$ to values determined directly by the centralized controller. This is one of the options available in "fixing the D-parameters".

The parameters over which the optimization takes place are then the α and β entries in the block canonical forms, the entries in all but the first columns of the matrices B^a_{aa} and B^e_{eea} and all the entries in the matrices $\begin{bmatrix} C^a_{aa} \\ C^a_{ea_c a} \end{bmatrix}$, C^e_{ee}, $D^a_{ea_c a}$ and D^e_{eea}. The parameter vector will be denoted as $\mathbf{p} \in R^N$ where $N = n_a(k_a + m_a + l_a + p_{ea}) + n_e(k_e + m_e + l_e + p_{ea}) + p_{ea}(k_a + m_e)$; n, m, k, and l refer respectively to the order, number of error inputs, number of outputs, and number of direct measurements for a subcontroller and p_{ea} refers to the number of interface variables. The number of parameters depends not only on the total numbers of controller inputs and outputs and interface variables which are fixed but also on the orders of the subcontrollers, n_a and n_e. There is thus a double incentive for keeping these orders low; not only to reduce the complexities of the subcontrollers but also to accelerate the optimization algorithm whose performance depends on the total number of parameters.

The Cost Function. The cost function is the sum of $f_0(\mathbf{p})$ as in (2.2) and $f_1(\mathbf{p})$ as in (2.3). These involve the transfer matrices $K(s)$, $\widetilde{K}(s)$, $T_{\text{cent}}(s)$, and $\widehat{T}(s)$ which are described below.

The parameters in the cost function are the entries in SKA and SKE as described in the previous section. These parameters along with some fixed transfer matrices are used in the determination of the transfer matrices needed in the cost function.

State-space representations for the centralized controller transfer matrix, $K(s)$, and the plant transfer matrix, $\widehat{G(s)} = \begin{bmatrix} G(s) \\ G_{ea}(s) \end{bmatrix}$, are given. Those for $\widetilde{K}(s)$, $T_{\text{cent}}(s)$ and $\widehat{T}(s)$ (shown in formulas (I.2), (I.3) and (I.4) below) are constructed from the state-space representations of $K(s)$, $K^a(s)$, $K^e(s)$, $G_{ea}(s)$ and $G(s)$.

The transfer matrix $\widetilde{K}(s)$ which enters into the performance cost term $f_0(\mathbf{p})$ depends on $K^a(s)$ and $K^e(s)$, and on the transfer submatrix of the plant from control inputs ($\mathbf{u}$) to interface variables ($\mathbf{z}_{ea}$), G_{ea} : $\mathbf{z}_{ea} = C^p_{eap}(sI - A^p)^{-1}\left[B^p_{pa}\ B^p_{pe}\right]\begin{bmatrix}\mathbf{u}_a \\ \mathbf{u}_e\end{bmatrix}$. The block diagram in Fig. I.1 shows the specific interconnections accounted for in this transfer matrix. Note that $\widetilde{K}(s)$ has the same inputs and outputs as $K(s)$ as described in (2.1).

A state-space realization for the equivalent partitioned controller $\widetilde{K}(s) = \widetilde{C}(sI - \widetilde{A})^{-1}\widetilde{B} + \widetilde{D}$ was shown in [**10**] to be $\widetilde{S} = \begin{bmatrix} \widetilde{A} & \widetilde{B} \\ \widetilde{C} & \widetilde{D} \end{bmatrix}$ where

$$
\begin{aligned}
\widetilde{A} &= \begin{bmatrix} A^a & 0 & 0 \\ B^e_{eea}C^a_{ea_c a} & A^e & -B^e_{eea}C^p_{eap} \\ (B^p_{pa}C^a_{aa} + B^p_{pe}D^e_{eea}C^a_{ea_c a}) & B^p_{pe}C^e_{ee} & (A^p - B^p_{pe}D^e_{eea}C^p_{eap}) \end{bmatrix} \\
\widetilde{B} &= \begin{bmatrix} B^a_{aa} & 0 & B^a_{ay_a} & 0 \\ B^e_{eea}D^a_{ea_c a} & B^e_{ee} & B^e_{eea}D^a_{ea_c y_a} & B^e_{ey_e} \\ (B^p_{pa}D^a_{aa} + B^p_{pe}D^e_{eea}D^a_{ea_c a}) & B^p_{pe}D^e_{ee} & (B^p_{pe}D^e_{eea}D^a_{ea_c y_a} + B^p_{pa}D^a_{ay_a}) & B^p_{pe}D^e_{ey_e} \end{bmatrix} \\
\widetilde{C} &= \begin{bmatrix} C^a_{aa} & 0 & 0 \\ D^e_{eea}C^a_{ea_c a} & C^e_{ee} & -D^e_{eea}C^p_{eap} \end{bmatrix} \\
\widetilde{D} &= \begin{bmatrix} D^a_{aa} & 0 & D^a_{ay_a} & 0 \\ D^e_{eea}D^a_{ea_c a} & D^e_{ee} & D^e_{eea}D^a_{ea_c y_a} & D^e_{ey_e} \end{bmatrix}
\end{aligned}
\qquad \text{(I.2)}
$$

The calculation of the tracking cost in (2.3) requires two transfer matrices, $T_{\text{cent}}(s)$ and $\widehat{T}(s)$. The norms of the rows of their difference measures the differences in response of the various possible interface variables to airframe commands. Figure I.2 is the block diagram for $T_{\text{cent}}(s) = C^{\text{cent}}(sI - A^{\text{cent}})^{-1}B^{\text{cent}} + D^{\text{cent}}$, the closed loop transfer matrix from airframe commands, $\mathbf{z}_{a_c}$, to interface variable quantities, $\mathbf{z}_{ea_c}$, produced by the engine using the centralized controller.

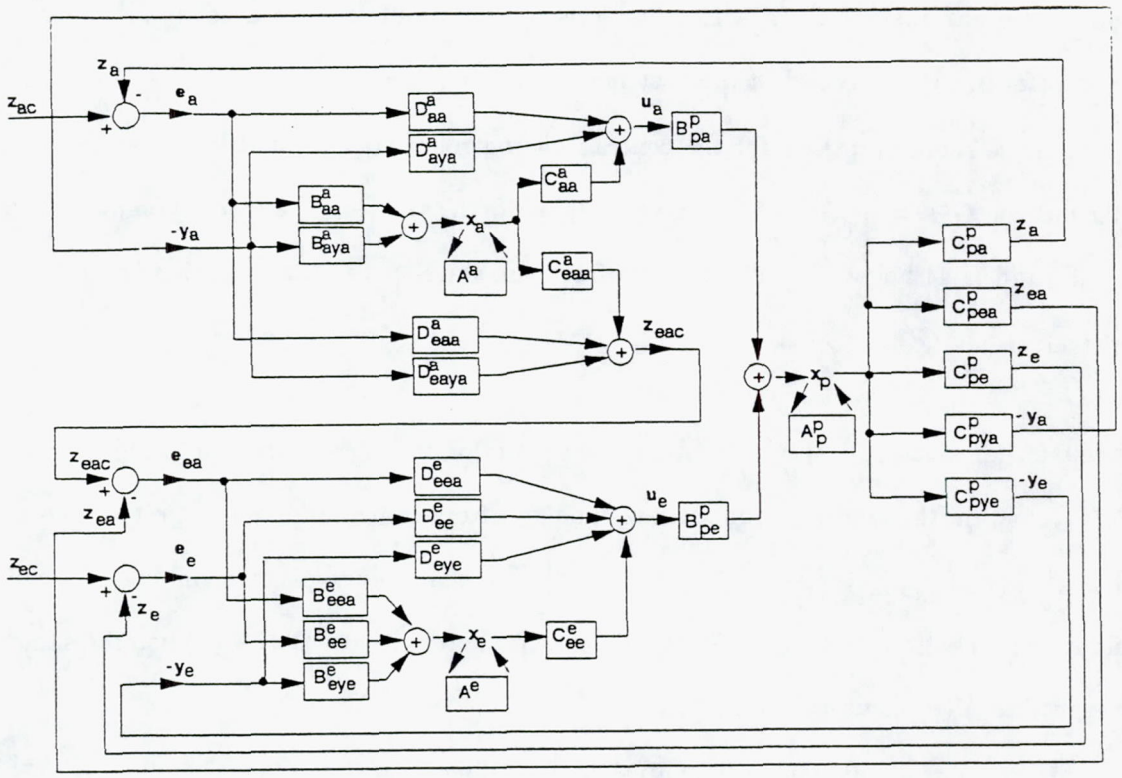
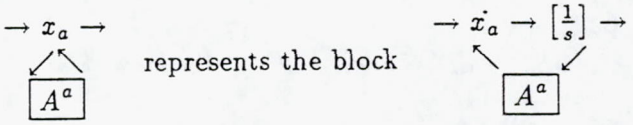


Figure I.1 Partitioned Controller

Note that in Figures I.1, I.2 and I.3 a block of the form



As is easily seen from Figure I.2 the state-space representation for the centralized controller transfer matrix $T_{\text{cent}}(s)$ can be written in terms of the submatrices in the state-space representations for $K(s)$ and $G(s)$,

$$A^{\text{cent}} = \begin{bmatrix} A^c & (-B^c_{ca}C^p_{ap} - B^c_{ce}C^p_{ep} + B^c_{cy_a}C^p_{y_ap} + B^c_{cy_e}C^p_{y_ep}) \\ (B^p_{pa}C^c_{ac} + B^p_{pe}C^c_{ec}) & A^p + B^p_{pa}(-D^c_{aa}C^p_{ap} + D^c_{ay_a}C^p_{y_ap} + D^c_{ae}C^p_{ep} + D^c_{ay_e}C^p_{y_ep}) \\ & + B^p_{pe}(-D^c_{ee}C^p_{ep} + D^c_{ey_a}C^p_{y_ap} + D^c_{ee}C^p_{ep} + D^c_{ey_e}C^p_{y_ep}) \end{bmatrix}$$

$$B^{\text{cent}} = \begin{bmatrix} B^c_{ca} \\ B^p_{pa}D^c_{aa} \end{bmatrix} \qquad C^{\text{cent}} = [\,0\ C^p_{eap}\,] \qquad D^{\text{cent}} = 0. \tag{I.3}$$

Figure I.3 is the block diagram for $\widehat{T}(s) = \widehat{C}(sI - \widehat{A})^{-1}\widehat{B} + \widehat{D}$, the closed loop transfer matrix from airframe commands, $\mathbf{z}_{a_c}$, to interface variable quantities produced by the airframe subcontroller, $\mathbf{z}_{ea_c}$ using the partitioned controller.

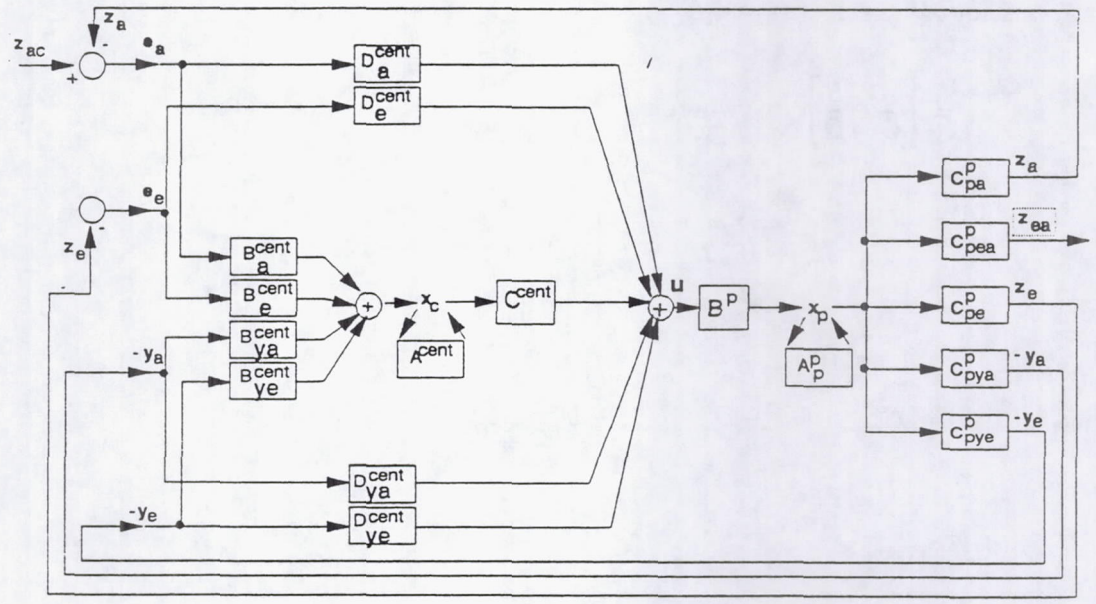


Figure I.2 z_{ea} response to z_{a_c} using the centralized controller

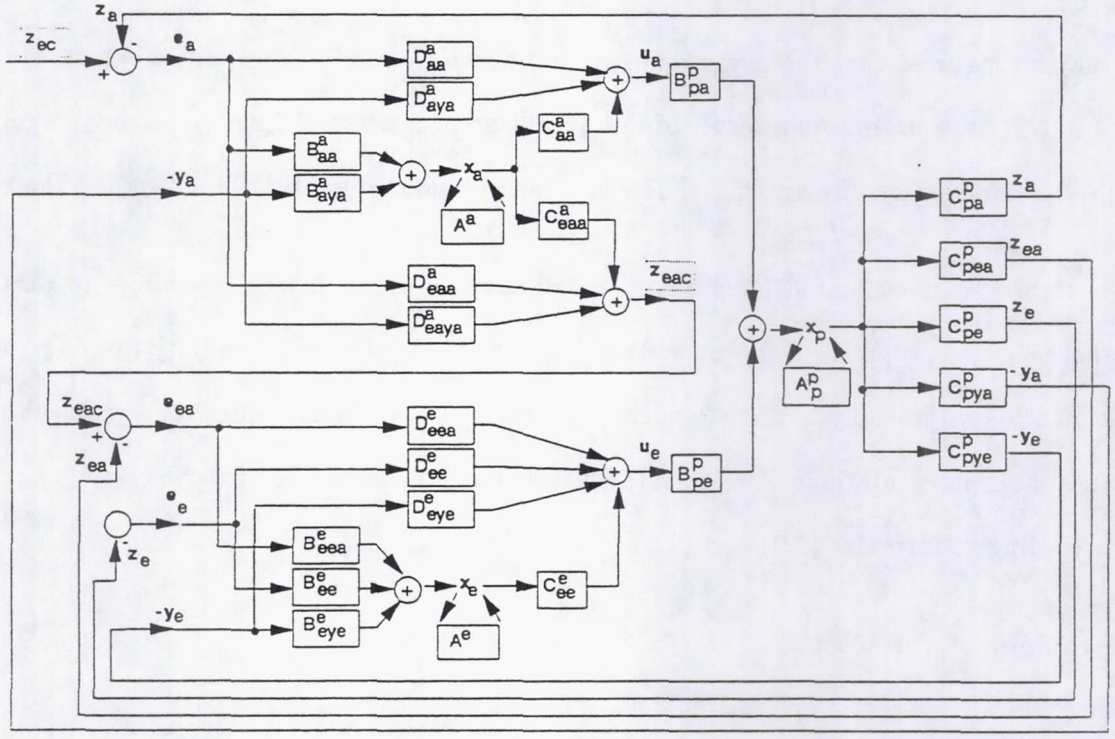


Figure I.3 z_{ea_c} response to z_{a_c} using the partitioned controller

The state space matrices for $\widehat{T}(s) = \widehat{C}(sI - \widehat{A})^{-1}\widehat{B} + \widehat{D}$ can be written in terms of the state-space representations for $K^a(s)$, $K^e(s)$ and $G(s)$ as

$$\widehat{T} = \begin{bmatrix} \widehat{A} & \widehat{B} \\ \widehat{C} & \widehat{D} \end{bmatrix} \qquad \text{where}$$

$$\widehat{A} = \begin{bmatrix} A^a & 0 & -B^a_{aa}C^p_{ap} + B^a_{ay_a}C^p_{y_ap} \\ B^e_{eea}C^a_{ea_ca} & A^e & \widehat{A}_{ep} \\ B^p_{pa}C^a_{aa} + B^p_{pe}D^e_{eea}C^a_{ea_ca} & B^p_{pe}C^e_{ee} & \widehat{A}_{pp} \end{bmatrix}$$

$$\widehat{B} = \begin{bmatrix} B^a_{aa} \\ B^e_{eea}D^a_{ea_ca} \\ B^p_{pe}D^a_{aa} + B^p_{pe}D^e_{eea}D^a_{ea_ca} \end{bmatrix}$$

$$\widehat{C} = [\, C^a_{ea_ca} \quad 0 \quad -D^a_{ea_ca}C^p_{ap} + D^a_{ea_cy_a}C^p_{y_ap} \,]$$

$$\widehat{D} = [\, D^a_{ea_ca} \,]$$

(I.4)

where $\widehat{A}_{ep} = (-B^e_{ee}C^p_{ep} - C^e_{eea}C^p_{eap} - B^e_{eea}D^a_{ea_ca}C^p_{ap} + B^e_{eea}D^a_{ea_cy_a}C^p_{y_ap} + B^e_{ey_c}C^p_{y_cp})$ and $\widehat{A}_{pp} = (A^p - B^p_{pa}D^a_{aa}C^p_{ap} - B^p_{pe}D^e_{eea}D^a_{ea_ca}C^p_{ap} - B^p_{pe}D^e_{eea}C^p_{eap} - B^p_{pe}D^e_{ee}C^p_{ep} + B^p_{pa}D^a_{ay_a}C^p_{y_ap} + B^p_{pe}D^e_{eea}D^a_{ea_cy_a}C^p_{y_ap} + B^p_{pe}D^e_{ey_c}C^p_{y_cp})$.

It may be required that the engine subsystem be strictly proper, a condition which would be violated if D^e_{eea} is nonzero. As a result of the optimization process, D^e_{eea} may become large. This possibility is removed by "fixing $D^e_{eea} = 0$" when the option is presented while running the code.

The total cost is evaluated for a particular parameter vector $\mathbf{p}$ (corresponding to particular SKA and SKE) by applying Simpson's Rule for numerical integration to $f_0(\mathbf{p}) + f_1(\mathbf{p})$ over a user defined logarithmically spaced frequency interval $[\omega_1, \omega_2]$. The expressions given above for the state-space representations of $K(s), \widetilde{K}(s), T_{\text{cent}}(s)$ and $\widehat{T}(s)$ are used for calculating $f_0(\mathbf{p})$ and $f_1(\mathbf{p})$ according to formulas (2.2) and (2.3).

The Gradient of the Cost.

The performance cost $f_0(\mathbf{p})$ was defined in (2.2) as

$$f_0(\mathbf{p}) = \int_0^\infty \frac{1}{N_{\text{PERF}}} \text{tr}\left[\left(W_o(K - \widetilde{K(\mathbf{p})})W_i\right)^* \left(W_o(K - \widetilde{K(\mathbf{p})})W_i\right)\right] \mathrm{d}\omega$$

where the explicit dependence of the integrand on ω is suppressed for convenience.

Only $\widetilde{K(\mathbf{p})}$ depends on $\mathbf{p}$ so the derivative of $f_0(\mathbf{p})$ with respect to a parameter ρ of $\mathbf{p}$ is

$$\begin{aligned}\frac{\partial f_0(\mathbf{p})}{\partial \rho} &= \int_0^\infty \frac{1}{N_{\text{PERF}}} \text{tr}\Big[\frac{\partial}{\partial \rho}\Big(W_o(K - \widetilde{K(\mathbf{p})})W_i\Big)^* \Big(W_o(K - \widetilde{K(\mathbf{p})})W_i\Big) \\ &\qquad + \Big(W_o(K - \widetilde{K(\mathbf{p})})W_i\Big)^* \frac{\partial}{\partial \rho}\Big(W_o(K - \widetilde{K(\mathbf{p})})W_i\Big)\Big] \mathrm{d}\omega \\ &= -2 \text{ Re} \int_0^\infty \frac{1}{N_{\text{PERF}}} \text{tr}\left[\Big(W_o(K - \widetilde{K(\mathbf{p})})W_i\Big)^* W_o \frac{\partial \widetilde{K(\mathbf{p})}}{\partial \rho} W_i\right] \mathrm{d}\omega\end{aligned}$$

Since $\widetilde{K(\mathbf{p})}(s) = \widetilde{C}(sI - \widetilde{A})^{-1}\widetilde{B} + \widetilde{D}$, the product rule for differentiation implies that

$$\begin{aligned}\frac{\partial \widetilde{K(\mathbf{p})}(s)}{\partial \rho} &= \frac{\partial \widetilde{C}}{\partial \rho}(sI - \widetilde{A})^{-1}\widetilde{B} + \widetilde{C}(sI - \widetilde{A})^{-1}\frac{\partial \widetilde{A}}{\partial \rho}(sI - \widetilde{A})^{-1}\widetilde{B} + \widetilde{C}(sI - \widetilde{A})^{-1}\frac{\partial \widetilde{B}}{\partial \rho} + \frac{\partial \widetilde{D}}{\partial \rho} \\ &= \begin{bmatrix}\widetilde{C}(sI - \widetilde{A})^{-1} & I\end{bmatrix} \frac{\partial}{\partial \rho}\begin{bmatrix}\widetilde{A} & \widetilde{B} \\ \widetilde{C} & \widetilde{D}\end{bmatrix} \begin{bmatrix}(sI - \widetilde{A})^{-1}\widetilde{B} \\ I\end{bmatrix}\end{aligned}$$

Thus,

$$\begin{aligned}\frac{\partial f_0(\mathbf{p})}{\partial \rho} = -2 \text{ Re} \int_0^\infty \frac{1}{N_{\text{PERF}}} \text{tr}\Big[&\Big(W_o(K - \widetilde{K(\mathbf{p})})W_i\Big)^* \\ &W_o\begin{bmatrix}\widetilde{C}(j\omega I - \widetilde{A})^{-1} & I\end{bmatrix} \frac{\partial}{\partial \rho}\begin{bmatrix}\widetilde{A} & \widetilde{B} \\ \widetilde{C} & \widetilde{D}\end{bmatrix} \begin{bmatrix}(j\omega I - \widetilde{A})^{-1}\widetilde{B} \\ I\end{bmatrix} W_i\Big] \mathrm{d}\omega\end{aligned}$$

Using abbreviations for the system matrix given in (I.2) well as for the terms on the left and right sides of the partial,

$$\begin{aligned}\widetilde{S} &= \begin{bmatrix}\widetilde{A} & \widetilde{B} \\ \widetilde{C} & \widetilde{D}\end{bmatrix} \\ L(\omega) &= \frac{1}{N_{\text{PERF}}}\Big(W_o(K - \widetilde{K(\mathbf{p})})W_i\Big)^* W_o\begin{bmatrix}\widetilde{C}(j\omega I - \widetilde{A})^{-1} & I\end{bmatrix} \\ R(\omega) &= \begin{bmatrix}(j\omega I - \widetilde{A})^{-1}\widetilde{B} \\ I\end{bmatrix} W_i\end{aligned}$$

allows the derivative to be written as

$$\frac{\partial f_0(\mathbf{p})}{\partial \rho} = -2 \text{ Re} \int_0^\infty \text{tr}\left[L(\omega)\frac{\partial \widetilde{S}}{\partial \rho} R(\omega)\right] \mathrm{d}\omega.$$

Any particular parameter ρ of $\mathbf{p}$ is some jk^{th} entry of some submatrix, denoted $\mathcal{M}_{\mathbf{oi}}$, relating an input $\mathbf{i}$ to an output $\mathbf{o}$ of SKA (or SKE) as described in (I.1). Furthermore, ρ (as well as the entire submatrix $\mathcal{M}_{\mathbf{oi}}$) occurs in one or more blocks of $\widetilde{S} = \begin{bmatrix}\widetilde{A} & \widetilde{B} \\ \widetilde{C} & \widetilde{D}\end{bmatrix}$ as was described in (I.2).

$\widetilde{S}$ can be thought of as consisting of blocks aligned in "columns" corresponding to the "inputs" $\mathbf{x}_a$, $\mathbf{x}_e$, $\mathbf{x}_p$, $\mathbf{e}_a$, $\mathbf{e}_e$, $\mathbf{y}_a$, $\mathbf{y}_e$ and in "rows" corresponding to the "outputs" $\mathbf{x}_a$, $\mathbf{x}_e$, $\mathbf{x}_p$, $\mathbf{u}_a$, $\mathbf{u}_e$. Each block of $\widetilde{S}$ is denoted as $\mathcal{B}_{\mathbf{OI}}$ where $\mathbf{I}$ is one of the "inputs" above and $\mathbf{O}$ is one of the "outputs". Let $E_{\mathbf{O}}$ denote a column block matrix with the same number of rows as $\widetilde{S}$, the same number of columns as the dimension of the "output" $\mathbf{O}$, with an identity matrix in the rows corresponding to "output" $\mathbf{O}$, and with zeros elsewhere. Let ${E_{\mathbf{I}}}^T$ be a similar row block matrix corresponding to the "input" $\mathbf{I}$,

$$E_{\mathbf{O}} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ I \\ 0 \\ \vdots \\ 0 \end{bmatrix} \qquad {E_{\mathbf{I}}}^T = [0 \ \cdots \ 0 \ I \ 0 \ \cdots \ 0].$$

Using this notation, $\widetilde{S}$ can be written as

$$\widetilde{S} = \sum_{\mathbf{O}} \sum_{\mathbf{I}} E_{\mathbf{O}} \mathcal{B}_{\mathbf{OI}} {E_{\mathbf{I}}}^T.$$

In the partial derivative of $\widetilde{S}$ with respect to the $\rho = \mathcal{M}_{\mathbf{oi}jk}$, every block of $\widetilde{S}$ is zero except for the blocks containing ρ. Denote such a block as $\mathcal{B}_{\mathbf{OI}} = \mathcal{L}_{\mathbf{Oo}} \mathcal{M}_{\mathbf{oi}} \mathcal{R}_{\mathbf{iI}}$, where $\mathcal{L}_{\mathbf{Oo}}$ denotes the factor to the left of $\mathcal{M}_{\mathbf{oi}}$ (if one exists, otherwise an identity) and $\mathcal{R}_{\mathbf{iI}}$ denotes the factor to the right(similarly, an identity if $\mathcal{M}_{\mathbf{oi}}$ is the rightmost factor); define both $\mathcal{L}_{\mathbf{Oo}} = \mathbf{0}$ and $\mathcal{R}_{\mathbf{iI}} = \mathbf{0}$ if $\mathcal{B}_{\mathbf{OI}}$ does not contain $\mathcal{M}_{\mathbf{oi}}$. The partial of $\widetilde{S}$ with respect to ρ thus contains a term of the form $\mathcal{L}_{\mathbf{Oo}} \mathbf{e}_j {\mathbf{e}_k}^T \mathcal{R}_{\mathbf{iI}}$ in the same place as the block $\mathcal{B}_{\mathbf{OI}}$. These contributions can be written as

$$\frac{\partial \widetilde{S}}{\partial \rho} = \sum_{\mathbf{O}} \sum_{\mathbf{I}} E_{\mathbf{O}} \mathcal{L}_{\mathbf{Oo}} \mathbf{e}_j {\mathbf{e}_k}^T \mathcal{R}_{\mathbf{iI}} {E_{\mathbf{I}}}^T$$

Thus the partial of the partitioning cost can be expressed as

$$\frac{\partial f_0(\mathbf{p})}{\partial \rho} = -2 \ \mathrm{Re} \int_0^\infty \mathrm{tr} \left[L(\omega) \left(\sum_{\mathbf{O}} \sum_{\mathbf{I}} E_{\mathbf{O}} \mathcal{L}_{\mathbf{Oo}} \mathbf{e}_j {\mathbf{e}_k}^T \mathcal{R}_{\mathbf{iI}} {E_{\mathbf{I}}}^T \right) R(\omega) \right] \mathrm{d}\omega.$$

Note that $L(\omega)E_{\mathbf{O}}\mathcal{L}_{\mathbf{Oo}}\mathbf{e}_j$ is a column vector and ${\mathbf{e}_k}^T\mathcal{R}_{\mathbf{iI}}{E_{\mathbf{I}}}^T R(\omega)$ is a row vector. Furthermore, the trace of a column times a row is the dot product of the row and the column. Therefore,

$$\frac{\partial f_0(\mathbf{p})}{\partial \rho} = -2\ \mathrm{Re} \int_0^\infty \sum_{\mathbf{O}} \sum_{\mathbf{I}} \left[({\mathbf{e}_k}^T\mathcal{R}_{\mathbf{iI}}{E_{\mathbf{I}}}^T R(\omega))(L(\omega)E_{\mathbf{O}}\mathcal{L}_{\mathbf{Oo}}\mathbf{e}_j)\right] \mathrm{d}\omega.$$

To generate the partials with respect to all the parameters in $\mathcal{M}_{\mathbf{oi}}$ simultaneously, let j and k vary over the row and column indicies respectively of the submatrix. Notice that varying the row index j selects the j^{th} column of the product $L(\omega)E_{\mathbf{O}}\mathcal{L}_{\mathbf{Oo}}$, whereas varying the column index k selects the k^{th} row of $\mathcal{R}_{\mathbf{iI}}{E_{\mathbf{I}}}^T R(\omega)$.

A matrix containing the partial derivatives with respect to the entries of $\mathcal{M}_{\mathbf{oi}}$ located in each entry's proper spot is thus obtained by replacing $\mathbf{e}_j$ and $\mathbf{e}_k$ by identity matrices and transposing the result.

$$\frac{\partial f_0(\mathbf{p})}{\partial \mathcal{M}_{\mathbf{oi}}} = -2\ \mathrm{Re} \int_0^\infty \sum_{\mathbf{O}} \sum_{\mathbf{I}} \left[(\mathcal{R}_{\mathbf{iI}}{E_{\mathbf{I}}}^T R(\omega))(L(\omega)E_{\mathbf{O}}\mathcal{L}_{\mathbf{Oo}})\right]^T \mathrm{d}\omega$$

Note that only the terms in $R(\omega)L(\omega)$ depend on ω and thus the integration can be rewritten to yield

$$\frac{\partial f_0(\mathbf{p})}{\partial \mathcal{M}_{\mathbf{oi}}} = -2\ \mathrm{Re} \left[\sum_{\mathbf{O}} \sum_{\mathbf{I}} \mathcal{R}_{\mathbf{iI}}{E_{\mathbf{I}}}^T \left[\int_0^\infty [R(\omega)L(\omega)]\mathrm{d}\omega\right] E_{\mathbf{O}}\mathcal{L}_{\mathbf{Oo}}\right]^T$$

Finally, build two matrices denoted $\dfrac{\partial f_0(\mathbf{p})}{\partial \mathrm{SKA}}$ and $\dfrac{\partial f_0(\mathbf{p})}{\partial \mathrm{SKE}}$ of the shapes of SKA and SKE (described in (I.1)) respectively containing the partial derivatives of $f_0(\mathbf{p})$ with respect to the parameters in SKA (respectively SKE) in the same positions as the corresponding parameters would occur. This is done so that in the software the gradient vector can be produced from $\dfrac{\partial f_0(\mathbf{p})}{\partial \mathrm{SKA}}$ and $\dfrac{\partial f_0(\mathbf{p})}{\partial \mathrm{SKE}}$ by a call of the function LONGCOL (the same function which produces $\mathbf{p}$ from SKA and SKE).

To build these matrices, define the "row" and "column" block matrices ${E^a}_{\mathbf{i}}{}^T$ and ${E^a}_{\mathbf{o}}$ relative to the "inputs" $\mathbf{i} \in \{\mathbf{x}_a, \mathbf{e}_a, \mathbf{y}_a\}$ and "outputs" $\mathbf{o} \in \{\mathbf{x}_e, \mathbf{u}_a, \mathbf{z}_{ea_c}\}$ for the airframe controller state-space system matrix SKA. As before ${E^a}_{\mathbf{o}}$ is a block column matrix with as many rows as SKA, with an identity matrix of size equal to the dimension of the output $\mathbf{o}$ in the rows corresponding to $\mathbf{o}$ and with zeros in the remaining rows. The matrices column block matrix ${E^e}_{\mathbf{o}}$ and the row block matrices ${E^a}_{\mathbf{i}}{}^T$, and ${E^e}_{\mathbf{i}}{}^T$ are similarly defined. If we again denote the submatrices of SKA

as $\mathcal{M}^a{}_{\mathbf{oi}}$ and those of SKE as $\mathcal{M}^e{}_{\mathbf{oi}}$, we can write

$$\text{SKA} = \sum_{\mathbf{o}} \sum_{\mathbf{i}} E^a{}_{\mathbf{o}} \mathcal{M}^a{}_{\mathbf{oi}} E^a{}_{\mathbf{i}}{}^T \quad \text{and} \quad \text{SKE} = \sum_{\mathbf{o}} \sum_{\mathbf{i}} E^e{}_{\mathbf{o}} \mathcal{M}^e{}_{\mathbf{oi}} E^e{}_{\mathbf{i}}{}^T .$$

Notice that, as with $\widetilde{S}$, the column and row block matrices merely position each $\mathcal{M}_{\mathbf{oi}}$ properly. Thus, replacing $\mathcal{M}_{\mathbf{oi}}$ by the corresponding block $\dfrac{\partial f_0(\mathbf{p})}{\partial \mathcal{M}_{\mathbf{oi}}}$ in these formulas gives the desired results

$$\frac{\partial f_0(\mathbf{p})}{\partial \text{SKA}} = \sum_{\mathbf{o}} \sum_{\mathbf{i}} E^a{}_{\mathbf{o}} \frac{\partial f_0(\mathbf{p})}{\partial \mathcal{M}^a{}_{\mathbf{oi}}} E^a{}_{\mathbf{i}}{}^T$$

and

$$\frac{\partial f_0(\mathbf{p})}{\partial \text{SKE}} = \sum_{\mathbf{o}} \sum_{\mathbf{i}} E^e{}_{\mathbf{o}} \frac{\partial f_0(\mathbf{p})}{\partial \mathcal{M}^e{}_{\mathbf{oi}}} E^e{}_{\mathbf{i}}{}^T$$

Using the expression from above for $\dfrac{\partial f_0(\mathbf{p})}{\partial \mathcal{M}^a{}_{\mathbf{oi}}}$ gives

$$\begin{aligned}
\frac{\partial f_0(\mathbf{p})}{\partial \text{SKA}} &= -2 \,\text{Re} \sum_{\mathbf{o}} \sum_{\mathbf{i}} E^a{}_{\mathbf{o}} \left[\sum_{\mathbf{O}} \sum_{\mathbf{I}} \mathcal{R}_{\mathbf{iI}} E_{\mathbf{I}}{}^T \left[\int_0^\infty [R(\omega) L(\omega)] \mathrm{d}\omega \right] E_{\mathbf{O}} \mathcal{L}_{\mathbf{Oo}} \right]^T E^a{}_{\mathbf{i}}{}^T \\
&= -2 \,\text{Re} \left[\sum_{\mathbf{o}} \sum_{\mathbf{i}} \sum_{\mathbf{O}} \sum_{\mathbf{I}} E^a{}_{\mathbf{i}} \mathcal{R}_{\mathbf{iI}} E_{\mathbf{I}}{}^T \left[\int_0^\infty [R(\omega) L(\omega)] \mathrm{d}\omega \right] E_{\mathbf{O}} \mathcal{L}_{\mathbf{Oo}} E^a{}_{\mathbf{o}}{}^T \right]^T .
\end{aligned}$$

The terms of the form $E^a{}_{\mathbf{i}} \mathcal{R}_{\mathbf{iI}} E_{\mathbf{I}}{}^T$ are independent of "outputs" $\mathbf{o}$ and $\mathbf{O}$ whereas those of the form $E_{\mathbf{O}} \mathcal{L}_{\mathbf{Oo}} E^a{}_{\mathbf{o}}{}^T$ are independent of "inputs" $\mathbf{i}$ and $\mathbf{I}$. Therefore the sums can be rearranged

$$\frac{\partial f_0(\mathbf{p})}{\partial \text{SKA}} = -2 \,\text{Re} \left[\left(\sum_{\mathbf{i}} \sum_{\mathbf{I}} E^a{}_{\mathbf{i}} \mathcal{R}_{\mathbf{iI}} E_{\mathbf{I}}{}^T \right) \left(\int_0^\infty [R(\omega) L(\omega)] \mathrm{d}\omega \right) \left(\sum_{\mathbf{o}} \sum_{\mathbf{O}} E_{\mathbf{O}} \mathcal{L}_{\mathbf{Oo}} E^a{}_{\mathbf{o}}{}^T \right) \right]^T .$$

The sum $\sum_{\mathbf{i}} \sum_{\mathbf{I}} E^a{}_{\mathbf{i}} \mathcal{R}_{\mathbf{iI}} E_{\mathbf{I}}{}^T$ represents a matrix with "inputs" $\mathbf{I}$, the "inputs" of $\widetilde{S}$, and with "outputs" $\mathbf{i}$, the "inputs" for SKA. This matrix has the submatrix $\mathcal{R}_{\mathbf{iI}}$ as the block in the rows corresponding to $\mathbf{i}$ and the columns corresponding to $\mathbf{I}$; denote this matrix as $\mathbf{R}^a$. Similarly the matrix $\mathbf{L}^a = \sum_{\mathbf{o}} \sum_{\mathbf{O}} E_{\mathbf{O}} \mathcal{L}_{\mathbf{Oo}} E^a{}_{\mathbf{o}}{}^T$ contains the submatrix $\mathcal{L}_{\mathbf{Oo}}$ in its block with rows corresponding to the $\widetilde{S}$ "output" $\mathbf{O}$ and with columns corresponding to SKA "outputs" $\mathbf{o}$.

The following simple procedure can thus be used to determine $\dfrac{\partial f_0(\mathbf{p})}{\partial \text{SKA}}$:

1. For each block $\mathcal{B}_{\mathbf{OI}}$ of $\widetilde{S}$ containing a submatrix $\mathcal{M}^a{}_{\mathbf{oi}}$ from SKA, determine the left and right factors (or identity) $\mathcal{L}_{\mathbf{Oo}}$ and $\mathcal{R}_{\mathbf{iI}}$ of $\mathcal{M}^a{}_{\mathbf{oi}}$.

2. Enter $\mathcal{L}_{\mathrm{Oo}}$ in the appropriate block of $\mathbf{L}^a$ and $\mathcal{R}_{\mathrm{iI}}$ in the appropriate block of $\mathbf{R}^a$.

3. Calculate $\int_0^\infty R(\omega)L(\omega)\,\mathrm{d}\omega$ where

$$L(\omega) = \frac{1}{N_{\mathrm{PERF}}}\left(W_o(K - \widetilde{K(\mathbf{p})})W_i\right)^* W_o\left[\widetilde{C}(j\omega I - \widetilde{A})^{-1} \quad I\right]$$

$$R(\omega) = \begin{bmatrix}(j\omega I - \widetilde{A})^{-1}\widetilde{B} \\ I\end{bmatrix} W_i$$

4. Form

$$\frac{\partial f_0(\mathbf{p})}{\partial \mathrm{SKA}} = -2\ \mathrm{Re}\left[\mathbf{R}^a\ \left(\int_0^\infty [R(\omega)L(\omega)]\mathrm{d}\omega\right)\ \mathbf{L}^a\right]^T.$$

By a similar procedure

$$\frac{\partial f_0(\mathbf{p})}{\partial \mathrm{SKE}} = -2\ \mathrm{Re}\left[\mathbf{R}^e\ \left(\int_0^\infty [R(\omega)L(\omega)]\mathrm{d}\omega\right)\ \mathbf{L}^e\right]^T$$

where $\mathbf{L}^e$ and $\mathbf{R}^e$ contain the left and right factors respectively of $\mathcal{M}^e{}_{\mathrm{oi}}$ terms appearing in $\widetilde{S}$.

It is easy to use the representation of $\widetilde{S}$ as given in (I.2) to calculate

$$\mathbf{R}^a = \begin{bmatrix} I & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & I & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & I & 0 \end{bmatrix} \quad \text{and} \quad \mathbf{L}^a = \begin{bmatrix} I & 0 & 0 \\ 0 & 0 & B^e_{eea} \\ 0 & B^p_{pa} & B^p_{pe}D^e_{eea} \\ 0 & I & 0 \\ 0 & 0 & D^e_{eea} \end{bmatrix}$$

and

$$\mathbf{R}^e = \begin{bmatrix} I & 0 & 0 & 0 & 0 & 0 & 0 \\ C^e_{eea} & 0 & -C^p_{eap} & D^a_{ea_c a} & 0 & D^a_{ea_c y_a} & 0 \\ 0 & 0 & 0 & 0 & I & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & I \end{bmatrix} \quad \text{and} \quad \mathbf{L}^e = \begin{bmatrix} 0 & 0 \\ I & 0 \\ 0 & B^p_{pe} \\ 0 & 0 \\ 0 & I \end{bmatrix}.$$

The Tracking Cost

The tracking cost was given in (2.3) as

$$f_1(\mathbf{p}) = \int_0^\infty \sum_i \frac{\lambda_i}{N^i_{\mathrm{TRACK}}(\omega)}\left(\|T^i_{\mathrm{cent}} - \widehat{T^i}(\mathbf{p})\|_2\right)^2 \mathrm{d}\omega$$

where the state space representations for $T_{\mathrm{cent}}(s)$ and $\widehat{T}(s)$ are given in (I.3) and (I.4) respectively.

This function can be put into a form which is similar to $f_0(\mathbf{p})$ so the same procedure for calculating the gradient applies. Each row of the difference is normalized before the sum of squares is calculated. Note that the multiplication by non-negative λ_i and division by non-negative $N^i_{\text{TRACK}}(\omega)$ could also be absorbed into the normalized sum of squares by the use of their square roots. Multiplication (or division) of rows by factors can also be achieved by multiplication from the left by a diagonal matrix. The sum of squares of the row norms in the resulting product is the same as the sum of squares of all the entries or the H_2 norm of the diagonal weighted difference. In this case

$$f_1(\mathbf{p}) = \int_0^\infty \text{tr}\left[\left(\text{diag}\left(\sqrt{\frac{\lambda_i}{N^i_{\text{TRACK}}(\omega)}}\right)\left(T_{\text{cent}} - \widehat{T}(\mathbf{p})\right)\right)^* \text{diag}\left(\sqrt{\frac{\lambda_i}{N^i_{\text{TRACK}}(\omega)}}\right)\left(T_{\text{cent}} - \widehat{T}(\mathbf{p})\right)\right] \mathrm{d}\omega.$$

As before, denote

$$L_t(\omega) = \left(\text{diag}\left(\sqrt{\frac{\lambda_i}{N^i_{\text{TRACK}}(\omega)}}\right)\left(T_{\text{cent}} - \widehat{T}(\mathbf{p})\right)\right)^T \text{diag}\left(\sqrt{\frac{\lambda_i}{N^i_{\text{TRACK}}(\omega)}}\right)\left[\widehat{C}(j\omega I - \widehat{A})^{-1} \quad I\right]$$

$$R_t(\omega) = \begin{bmatrix}(j\omega I - \widehat{A})^{-1}\widehat{B} \\ I\end{bmatrix}$$

and apply the same procedure as earlier to write the partial derivatives as

$$\frac{\partial f_1(\mathbf{p})}{\partial SKA} = -2\,\text{Re}\left[\begin{bmatrix} I & 0 & 0 & 0 \\ 0 & 0 & -C^p_{ap} & I \\ 0 & 0 & C^p_{y_a p} & 0 \end{bmatrix} \int_0^\infty R_t(\omega)L_t(\omega)\mathrm{d}\omega \begin{bmatrix} I & 0 & 0 \\ 0 & 0 & B^e_{eea} \\ 0 & B^p_{pa} & B^p_{pe}D^e_{eea} \\ 0 & 0 & I \end{bmatrix}\right]^T$$

and

$$\frac{\partial f_1(\mathbf{p})}{\partial SKE} = -2\,\text{Re}\left[\begin{bmatrix} 0 & I & 0 & 0 & \\ C^e_{eea} & 0 & -C^p_{eap} - D^e_{eea}C^p_{eap} + D^a_{ea_c y_a}C^p_{y_a p} & 0 & D^a_{ea_c a} \\ 0 & 0 & -C^p_{ep} & 0 & \\ 0 & 0 & C^p_{y_c p} & 0 & \end{bmatrix} \int_0^\infty R_t(\omega)L_t(\omega)\mathrm{d}\omega \begin{bmatrix} 0 & 0 \\ I & 0 \\ 0 & B^p_{pe} \\ 0 & 0 \end{bmatrix}\right]^T$$

This completes the discussion of the cost function and its gradient as implemented in this software. If the user wishes to compute cost functions involving the H_2 norm as used here, the user must apply the procedure described above to determine the gradient.

APPENDIX II

SHORT DESCRIPTIONS OF USER-DEFINED FUNCTIONS

START in START.MTX

[SKA_OPT,SKA_OPT] = START(LAMBDA, STOP)

Initializes information from data files, puts subcontrollers into modified modal form, constructs or initializes some matrices required for the evaluation of the cost function. Calls COST to initialize the costs and gradients and then calls PARTITION to perform the optimization.

INPUT

LAMBDA — weighting for the contribution of the tracking cost to the total cost.

STOP (optional) — vector of stopping conditions.

INIT.DAT — data file containing initial information.

OUTPUT

SKA_OPT — the state-space representation of the optimized airframe subcontroller.

SKE_OPT — the state-space representation of the optimized engine subcontroller.

CONST.DAT — data file containing constants used by other UDFs.

PAR.DAT — used to store the stopping conditions in STOP.

RESTART in RESTART.MTX

[SKA_OPT,SKA_OPT] = RESTART(LAMBDANEW, STOP, NEWH)

Restarts the program using the data available in INTER.DAT and CONST.DAT. Calls COST to initialize the costs and gradients, and then calls PARTITION to optimize.

INPUT

LAMBDANEW — a new value of LAMBDA may allow a different emphasis on the tracking cost relative to the total cost.

STOP (optional) — the vector of stopping conditions, same as in the START routine. It may be redefined, perhaps to allow more stringent conditions or more iterations.

NEWH (optional) — a flag whose presence indicates the desire for restart with identity inverse Hessian. If restart with the current Hessian matrix is desired, no value should be passed.

CONST.DAT — data file containing constants pertaining to the plant and global controller being partitioned.

INTER.DAT — the data file in which intermediate results from previous iterations are stored.

OUTPUT

SKA_OPT — the state-space representation of the optimized airframe subcontroller.

SKE_OPT — the state-space representation of the optimized engine subcontroller.

PAR.DAT — the data file in which the stopping conditions STOP are saved.

INTER.DAT — the data file in which intermediate results are stored.

PARTITION in PARTITIO.MTX

P_I = PARTITION(STOP)

This is the main routine and does the optimization. It iterates till some convergence or stopping conditions are met. After each iteration a graph is plotted showing the change in costs. PARTITION implements the Broyden-Fletcher-Goldfarb-Shanno method of determining a search direction for minimization. It then calls the function INACCURATE to implement Fletcher's inaccurate line search in that direction. The function CONVERGE is called to see if convergence or stopping conditions are met; if not, the iteration in PARTITION is repeated.

INPUT

STOP — the vector of stopping conditions, used to check convergence.

CONST.DAT — the data file containing the constants of the program.

INTER.DAT — the data file containing the intermediate results.

OUTPUT

P_I — the vector of parameters, after the i^{th} optimization step.

INTER.DAT — the data file containing the intermediate results, where the results of the optimization are stored.

INACCURATE in LINESRCH.MTX

[ALI, FAL, FP0, FP1, DFDPAL, DFP0, DFP1, FLAG]

= INACCURATE (X, FZ, DFDPZ, D, FMIN, DELTAF, LAMBDA, ALPHAMAX)

This function performs Fletcher's inaccurate line search as part of the unconstrained optimization performed by the function PARTITION. It calls the function COST to get the cost and gradient for the cost function evaluated at the i^{th} set of parameters. Effectively, INACCURATE seeks a point where a sufficient decrease in both the function value and directional derivative have occurred.

INPUT

X — the current point (parameter vector), before the linesearch.

FZ — the total cost function evaluated at X.

DFDPZ — the gradient of the cost function evaluated at X.

D — the direction vector which the PARTITION function has chosen to perform the line search.

FMIN — the minimum value for the cost function. If the cost falls below this value, the function will terminate.

DELTAF — the estimated change in cost.

LAMBDA — the weighting for the tracking cost.

ALPHAMAX — maximum alpha, set by stability constraints.

OUTPUT

ALI — the alpha value that yields sufficient reduction in $F(X + alpha \times D)$ for the i^{th} iteration.

FAL — the total cost function evaluated at $X + AL \times D$.

FP0 — the partitioning cost evaluated at $X + AL \times D$.

FP1 — the tracking cost evaluated at $X + AL \times D$.

DFDPAL — the gradient of the total cost function evaluated at $X + AL \times D$.

DFP0 — the gradient of the partitioning cost at $X + AL \times D$.

DFP1 — the gradient of the tracking cost at $X + AL \times D$.

FLAG — an output flag signalling the condition with which INACCURATE completed.

0 — solution found, no problems.

1 — solution found, objective value less than FMIN.

2 — new point found, but backeting step converged to right endpoint without satisfying Armijo/Goldstein conditions.

COST in COST.MTX

[DF, DF0, DF1, FP, FP0, FP1] = COST (P, LAMBDA)

Evaluates the cost and gradient at the current point using the cost function described in Appendix I. The state space representations for $\widetilde{K}(s)$ and $\widehat{T}(s)$ are constructed based on the current value of the parameter P. The contributions of these to the costs f_0 and f_1 are calculated at each frequency point in OMEGA and are added according to Simpson's Rule for numerical integration. Furthermore, the contributions to the gradient as described in Appendix I are also accumulated for each frequency in OMEGA.

INPUT

P — the current point (parameter vector).

LAMBDA — the weighting for the tracking cost.

CONST.DAT — the data file containing the constants of the program.

OUTPUT

DF, DF0, DF1 — the gradients of the total cost function, the partitioning cost function and the tracking cost function respectively.

FP, FP0, FP1 — the values of the total cost function, the partitioning cost function and the

tracking cost function respectively.

CONVERGE in CONVERGE.MTX

ANSWER = CONVERGE (X_I1, F_I1, X_I, F_I, GRAD, EPS, DEL, ETA)

This function checks if the convergence condition are met and returns a flag ANSWER defined as

$$\text{ANSWER} = \begin{cases} 0 & \text{if MAX } |X\_I1 - X\_I| > \text{EPS or } |F\_I1 - F\_I| > \text{DEL} \\ 1 & \text{if MAX } |X\_I1 - X\_I| < \text{EPS and } |F\_I1 - F\_I| < \text{DEL but MAX } |\text{GRAD}| \geq \text{ETA} \\ 2 & \text{if MAX } |X\_I1 - X\_I| < \text{EPS, } |F\_I1 - F\_I| < \text{DEL and MAX } |\text{GRAD}| < \text{ETA} \end{cases}$$

INPUT

X_I1 — the value of the variable (parameter vector) at the $(i+1)^{\text{th}}$ iteration.

F_I1 — the value of the cost function at the $(i+1)^{\text{th}}$ iteration.

X_I — the value of the variable (parameter vector) at the i^{th} iteration.

F_I — the value of the cost function at the i^{th} iteration.

GRAD — the gradient of the cost function at the $(i+1)^{\text{th}}$ iteration.

EPS — the absolute tolerance for X differences.

DEL — the absolute tolerence for cost function differences.

ETA — the absolute tolerence for the gradient.

OUTPUT

ANSWER — a flag indicating the state of convergence.

0 — no convergence.

1 — variable values converge, function values converge.

2 — variable values, function values, and gradient converge.

MODL in MODL.MTX

[SM]=MODL(S,NS)

This function takes a system matrix and puts it in to the form where A has 2×2 companion matrix blocks whose first rows are [0 1] and whose second rows are [a b]. The transformation matrix is

normalized by requiring that all nonzero entries in the first column of the B matrix remain fixed (zero entries are set at 10^{-9}).

INPUT

S — the system matrix.

NS — the order of the input matrix S.

OUTPUT

SM — the system matrix in modified modal form, still having order NS.

MAT in PARMAT.MTX

[SKA,SKE] = MAT(P)

Creates the system matrices for the airframe and the engine controllers from the parameter vector (the variable over which the optimization process is being performed). In case one of the subsystems is fixed, the corresponding initial system matrix is loaded from CONST.DAT and returned as SKA or SKE.

INPUT

P — the parameter vector.

CONST.DAT — the data file of constants.

OUTPUT

SKA — the system matrix for the airframe controller.

SKE — the system matrix for the engine controller.

LONGCOL in PARVEC.MTX

[P] = LONGCOL(SKA,SKE)

This function generates the parameter vector from the system matrices for the airframe and the engine. In case one of the subcontrollers is fixed, the parameter vector corresponds only to the

parameters in the other subcontroller.

INPUT

SKA — the airframe controller.

SKE — the engine controller.

CONST.DAT — the data file of constants.

OUTPUT

P — the long column vector of the parameters.

Z in ZERO.MTX

[ZER] = Z(NROW, NCOL)

Constructs a matrix of zeros of size NROW × NCOL.

INPUT

NROW, NCOL — the row and column size of the desired zero matrix.

OUTPUT

ZER — the generated zero matrix.

Appendix III

Controller Partitioning Code

```
// answer=converge(x_i1,f_i1,x_i,f_i,grd,epsilon,delta,eta,fx)
//
// The function CONVERGE located in the file CONVERGE.MTX  is called by
// PARTITION to test the convergence of the optimization algorithm according
// to the following criteria:
// Condition X: | xi1(j)-xi(j) | < epsilon(j) for j=1..n where dim(x)=n x 1
//   whether the parameter vectors are converging
// Condition f: | f(xi1)-f(xi) | < delta
//   whether the function values (costs) are converging
// Condition G: |grd(j)| < eta   for j=1..n where dim(x)=n x 1
//   whether the gradient is approaching some low value (approx 0)
// The result of the convergence check are returned in the flag answer.
// The codes used for the answer are:
//   answer = 1 if X and f are true and G is false.
//   answer = 2 if X,f and G are true
//   answer = 0 if either X or f is false
// Input:
//   x_i1 - [real,vec] value of variable at iteration (i+1).
//   x_i - [real,vec] value of variable at iteration (i).
//   f_i and f_i1 - [real,scal] values of function at successive iterations
//   g_i and g_i1 - [real,vec] gradients at successive iterations
//   epsilon - [real,vec] vector of the absolute tolerances defined
//     for each of the elements of the x
//   delta - [real,scal] absolute tolerance for the objective values.
//   eta - [real,scal] absolute tolerance for gradient
//   fx - [real,vec] tracks which of the A matrices is at its stability bound
// Output:
//   answer - [int,scal] flag that will show whether or not the
//     optimization procedure has converged according to the
//     user-defined criteria.
//
answer=0;
[row,col]=size(grd);

if abs(f_i1-f_i) > delta then retf;
diff=abs(x_i1-x_i);
for i=1:row; ...
  if diff(i)>epsilon(i), retf;...
end;

answer=1;
for i=1:row; ...
 if fx(i)=0, if grd(i)>eta, retf; ...
end;end;

answer=2; retf;


retf

//  Created:  01/24/90
//  Programmer:  Steven Ims
//  Revised by Phil Schmidt and students Nader Kamrani and Brian Holawecky at
//  The University of Akron with the support of NASA Lewis Research Center
//  under grant NAG-3-1146.
```

```
// [Df,Df0,Df1,fp,fp0,fp1]=cost(p,lambda)
//
// The function COST in file COST.MTX
// computes the cost and gradient of the cost function at the current
// point (parameter vector).  The partitioning and tracking costs and
// their gradients are returned seperately.
// Input:
//   p - the parameter vector
//   lambda - the weighting applied to the tracking cost in the total
//     cost function
//   const.dat - datafile of constants
// Output:
//   Df - gradient of the total cost function
//   Df0 - gradient of the partitioning cost function
//   Df1 - gradient of the tracking cost function
//   fp - the total cost at point p (current parameter vector)
//   fp0 - value of the partitioning cost function
//   fp1 - value of the tracking cost function
//

load 'const.dat' la le ka ma ke me pea nka nke sp sk np nk frq omega ...
  sg ng swi nwi swo nwo weight stabil aore Nperf Ntrack;

//Generate ska and ske controllers from the parameter vector p
[ska,ske]=mat(p);

//Note some of the sizes of matrices, numbers of inputs and outputs
mt=ma+me;        //ma,me - airframe, engine inputs
lt=la+le;        //la,le - numbers of airframe and engine y feedbacks
mtot=mt+lt;      //mtot - total numbers of inputs to the centrallized controller
kt=ka+ke;        //ka,ke - airframe and engine outputs
nt=np+nka+nke;   //np,nka,nke - orders of plant,airframe,engine controllers

// Split the plant controller sp and find the sizes of each piece
// using the stored data in init.dat.  Get the pieces from each submatrix M
// which connect input i to output o, that is Moi; Bpa,Bpe,Cap,Cyep
// Some of the indices into the rows and columns used repeatedly are
// precalculated to save processing, ie, mapl=ma+1 etc
[ap,bp,cp,Df]=split(sp,np);      //split sp
[mp ntmp]=size(cp);              //the ap: xp -> xp and does not need
mapl=ma+1;mame=ma+me;            // to be further broken up
mpea=ma+me+pea;                  //the dp submatrix of the plant is
kapl=ka+1; kp=ka+ke;             //kept zero
bpa=bp(:,[1:ka]);                //part of bp: ua -> xp
bpe=bp(:,[kapl:kp]);             //part of bp: ue -> xp
cap=cp([1:ma],:);                //part of cp: xp -> za
ceap=cp([mapl:ma+pea],:);        //part of cp: xp -> zeap
cep=cp([(ma+pea+1):mpea],:);     //part of cp: xp -> ze
cyap=cp([(mpea+1):(mpea+la)],:);//part of cp: xp -> ya
cyep=cp([(mpea+la+1):(mp)],:);   //part of cp: xp -> ye

// From the airframe controller ska, get Baa,Baya,Caa,Ceaa,
// Daa,Daya,Deaa,Deaya in similar manner to the way that the plant
// matrix was split up
[aa,ba,ca,da]=split(ska,nka);    //split the airframe controller
[katot ntmp]=size(ca);           //part of
baa=ba(:,[1:ma]);                //  ba: ea -> xa
baya=ba(:,[(ma+1):(ma+la)]);     //  ba: ya -> xa
caa=ca([1:ka],:);                //  ca: xa -> ua
ceaa=ca([kapl:katot],:);         //  ca: xa -> zea
daa=da([1:ka],[1:ma]);           //  da: ea -> ua
daya=da([1:ka],[mapl:(ma+la)]); //  da: ya -> ua
deaa=da([kapl:katot],[1:ma]);    //  da: ea -> zea
deaya=da([kapl:katot],[mapl:(ma+la)]); //  da: ya -> zea

// From the engine subcontroller, get Beea,Bee,Beye,Cee,Deeam,Deem,Deye
[ae,be,ce,de]=split(ske,nke);    //split the engine controller
[ntmp metot]=size(be);           //part of
beea=be(:,[1:pea]);              //  be: zea -> xe
bee=be(:,[(pea+1):(me+pea)]);    //  be: ee -> xe
beye=be(:,[(me+pea+1):metot]);   //  be: ye -> xe
cee=ce;                          //  ce: xe -> ue
deea=de(:,[1:pea]);              //  de: zea -> ue
dee=de(:,[(pea+1):(me+pea)]);    //  de: ee -> ue
deye=de(:,[(me+pea+1):metot]);   //  de: ye -> ue

//PARTITIONING COST MATRICES
// Get the Centralized controller K for the performance cost as
// input data originally read from init.dat
[ak,bk,ck,dk]= split(sk,nk);

// Form the Equivalent System tilde(K) using Partitioned Subcontrollers
// from inputs: Ee,Ea,Y to outputs: Ua,Ue.
at0=[aa,0*ones(nka,nke),0*ones(nka,np);..
  beea*ceaa,ae,-beea*ceap;..
  bpa*caa+bpe*deea*ceaa,bpe*ce,(ap-bpe*deea*ceap)];
bt0=[baa,0*ones(nka,me),baya,0*ones(nka,le);..
  beea*deaa,bee,beea*deaya,beye;..
  bpa*daa+bpe*deea*deaa,bpe*dee,bpe*deea*deaya+bpa*daya,bpe*deye];
ct0=[caa,0*ones(ka,nke),0*ones(ka,np);deea*ceaa,ce,(-deea*ceap)];
dt0=[daa,0*ones(ka,me),daya,0*ones(ka,le);..
  deea*deaa,dee,deea*deaya,deye];

//TRACKING COST MATRICES
// Form  Tcent from input: Zac  to  output: Zeac for
// command tracking using Centralized  Controller.  This was created in
// start.mtx because it only needs to be defined once (does not vary
// with the parameter)
[ag,bg,cg,dg]=split(sg,ng);

// Generate the state-space representation tilde(T) from input:Zac
// to output: Zeac, command tracking using Partitioned Controller.
atc=[aa,0*ones(nka,nke),(-baa*cap+baya*cyap);...
 beea*ceaa,ae,(-1*bee*cep-beea*ceap-beea*deaa*cap ...
 +beea*deaya*cyap+beye*cyep);...
 (bpa*caa+bpe*deea*ceaa), bpe*cee, (ap-bpa*daa*cap...
 -bpe*deea*deaa*cap-bpe*deea*ceap-bpe*dee*cep ...
 +bpa*daya*cyap+bpe*deea*deaya*cyap+bpe*deye*cyep)];
btc=[baa;beea*deaa;(bpa*daa+bpe*deea*deaa)];
ctc=[ceaa,0*ones(pea,nke),(-deaa*cap+deaya*cyap)];
dtc=[deaa];

// Decompose the input and output weighting functions into
// their state space representations
[awi,bwi,cwi,dwi]=split(swi,nwi);
[awo,bwo,cwo,dwo]=split(swo,nwo);

// Compute the costs; fp0 (performance), fp1 (tracking) and their gradients
//  fp0=Integral (tr ((Wo(K- tilde(K))Wi)^T) (Wo(K-tilde(K))Wi)) )
//  fp1=Integral (tr ((nrm(T- tilde(T)))^T) (nrm(T-tilde(T))))
//  Dfp0=partitioning cost gradient
//     =-2*Re[ (LeftFactor)
//       Integral
//        ( [inv(sI-A)*B; I]*Wi*((Wo(K- tilde(K))Wi)^T)*Wo*[C*inv(sI-A), I] )
//       (RightFactor) ]'
//  Dfp1=tracking cost gradient
//     =-2*Re[ (LeftFactor)
//       Integral
```

```
//        ( [inv(sI-A)*B; I]*(nrm*(T- tilde(T))^T*nrm*[C*inv(sI-A), I] )
//       (RightFactor) ]'

// Initialize the costs and gradients to zero
fp0=0;
fp1=0;
Df0=0*ones(nka+nke+np+ma+me+la+le,nka+nke+np+ka+ke);
Df1=0*ones(nka+nke+np+ma,nka+nke+np+pea);

// Integrate over the frequency interval [FRQ(1),FRQ(2)] using Simpsons rule,
// coefficients are in the variable WEIGHTS..
// Build up the partitioning costs fp0, fp1, and the integral
// part of the gradients Df0 and Df1
// Df0=Integral
//       ( [inv(sI-A)*B; I]*Wi*((Wo(K- tilde(K))Wi)^T)*Wo*[C*inv(sI-A), I] )
// Dfp1=Integral ( [inv(sI-A)*B; I]*(nrm*(T- tilde(T))^T*nrm*[C*inv(sI-A), I] )
for i=1:frq(3,1), om=omega(i);...
 k0=inv(om*jay*eye(nt)-at0); ...
 wi=cwi*inv(om*jay*eye(nwi)-awi)*bwi+dwi;...
 wo=cwo*inv(om*jay*eye(nwo)-awo)*bwo+dwo;...
 kdiff=wo*(ck*inv(om*jay*eye(nk)-ak)*bk+dk-ct0*k0*bt0-dt0)*wi;...
 fp0=fp0 +weight(i)*om*sum(diag(kdiff'*kdiff)/Nperf(i));...
 Df0=Df0+weight(i)*om/Nperf(i)*[k0*bt0;eye(mtot)]*wi*kdiff'*wo*...
  [ct0*k0,eye(kt)];...
 nrm=diag((lambda./Ntrack(i,:))**.5);...
 k1=inv(om*jay*eye(nt)-atc); ...
 ktrc=nrm*(cg*inv(om*jay*eye(ng)-ag)*bg+dg-ctc*k1*btc-dtc);...
 fp1=fp1+weight(i)*om*sum(diag(ktrc'*ktrc));...
 Df1=Df1+weight(i)*om*[k1*btc;eye(ma)]*ktrc'*nrm*[ctc*k1, eye(pea)];...
end;

//If the airframe controller is to be optimized, calculate the cost gradients
//with respect to the parameters belonging to the airframe controller by
//multiplying DF0 and Df1 by the corresponding Left and Right Factors
//This is done for both the partitioning cost Dfa0 and the tracking
//cost Dfe0 (with respect to the airframe parameters)
//Dfa0=[ (LeftFactor) Df0 (RightFactor) ]'
//Dfa1=[ (LeftFactor) Df1 (RightFactor) ]'
if aore<>1,...
Dfa0=([eye(nka),z(nka,nke+np+mtot);z(ma,nt),eye(ma),z(ma,me+lt);...
 z(la,nt+mt),eye(la),z(la,le)]'*Df0*[eye(nka),z(nka,ka+pea);z(nke,nka+ka)...
 beea;z(np,nka),bpa,bpe*deea;z(ka,nka),eye(ka),z(ka,pea);...
 z(ke,nka+ka),deea])';..
Dfa1=([eye(nka),z(nka,nke+np+ma);z(ma,nka+nke),-cap,eye(ma);z(la,nka+nke),...
 cyap,z(la,ma)]'*Df1*[eye(nka),z(nka,ka+pea);z(nke,nka+ka),beea;z(np,nka),...
 bpa,bpe*deea;z(pea,nka+ka),eye(pea)])';..
else Dfa0=0*ones(ska); Dfa1=Dfa0; end,

//Similarly, if the engine controller is to be optimized,
//calculate the cost gradients with respect to the engine
//parameters by multiplying Df0 and Df1 by the requisite Left and
//Right Factors.  Gradient of partitioning cost with respect to the
//engine parameters is Dfe0 and gradient of the tracking cost is Dfe1.
//Dfe0=[ (LeftFactor) Df0 (RightFactor) ]'
//Dfe1=[ (LeftFactor) Df1 (RightFactor) ]'
if aore<>2,...
Dfe0=([z(nke,nka),eye(nke),z(nke,np+mtot);ceaa,z(pea,nke),-ceap, deaa,...
 z(pea,me),deaya,z(pea,le);z(me,nt+ma),eye(me),z(me,lt);z(le,nt+mt+la),..
 eye(le)]...
*Df0*[z(nka,nke+ke);eye(nke),z(nke,ke);z(np,nke),bpe;z(ka,nke+ke);z(ke,nke),...
 eye(ke)])';...
Dfe1=([z(nke,nka),eye(nke),z(nke,np+ma);ceaa,z(pea,nke),...
 -(deaa*cap+ceap-deaya*cyap),deaa;z(me,nka+nke),-cep,z(me,ma);z(le,nka+nke),..
 cyep,z(le,ma)]'*Df1*[z(nka,nke+ke);eye(nke),z(nke,ke);z(np,nke),bpe;..
 z(pea,nke+ke)])';...
else Dfe0=0*ones(ske);Dfe1=Dfe0;end,

//Assemble the final cost and gradients using the formulas
//described earlier and the airframe and engine costs and gradients
//just generated.
coef=(log(frq(2,1)/frq(1,1)))/(3*frq(3,1));
fp0=real(coef*fp0); fp1=real(coef*fp1);
fp=fp0+fp1;
Df0=-2*coef*real(longcol(Dfa0,Dfe0));
Df1=-2*coef*real(longcol(Dfa1,Dfe1));
Df=Df0+Df1;

retf

// COST evaluates the total cost of approximating a centralized controller by
// hierarchically partitioned subcontrollers.  It is the sum of two costs,
// fPerf which measures the performance cost of approximation and fTrack which
// measures the error in meeting the command tracking requirement.  It's input
// is the parameter vector. p, and the tracking weighting vector, lambda.

// The file 'const.dat' contains the constants of the process
// the dimensions --- ka,ma,ke,me,pea,np,nka,nke,nk,nwi(o),ng,la,le(k=outputs,
//   m=inputs, l= neg. feedbacks, n=order, p=intermediate variables)
// the range vector of frequencies --- frq
// the actual frequencies --- omega
// the plant in system form --- sp
// the global controller in system form --- sk
// the input and output weighting matrices in system form --- swi, swo
// the global tracking command matrix in system form --- sg
// the weights used in the numerical integration --- weight
// the normalization matrix for the performance cost --- Nperf
// the normalization matrix for the tracking cost -- Ntrack
//
// The state space representation for the weighted difference between global
// controller and assembled partitioned controller is determined.  The
// H_2 norm of the resulting system is computed.The norm is calculated by
// applying a Simpson's rule suitably modified to account for the exponential
// distribution of the omegas.
//
// The normalized H_2 norm of the difference between the tracking command
// transfer matrix and the nominal one is also calculated.  This is added to
// the previous H_2 norm. The gradient of this sum is computed.
//
// The outputs are fp0 --- the H_2 norm of the weighted difference between
//                         centralized and partitioned  controllers
//                 fp1 --- the normalized H_2 norm  of the deviation from
//                         nominal command tracking
//                 fp = fp0 + fp1
// and Df0,Df1 and Df --- the gradients of fp1, fp1 and fp resp.


// Created by Phil Schmidt and students Nader Kamrani and Brian Holawecky at
// The University of Akron with the support of NASA Lewis Research Center
// under grant NAG-3-1146.
```

```
// [ali,fal,fp0,fp1,dfdpal,dfp0,dfp1,flag]=INACCURATE(x,fz,dfdpz,...
d,fmin,deltaf,lambda,alphamax)
//
// The function INNACURATE in file LINESRCH.MTX
// performs Fletchers inaccurate line search as part
// of the unconstrained optimization process.  Given the search direction,
// a point is found that satisfies the Armijo-Goldstein conditions using
// a two stage bracketing/sectioning procedure.
// Input:
//   x - [real,vec] the current point corresponding to al=0
//   fz - [real,scal] objective function  value for al=0.
//   dfdpz - [real,vec] directional derivative of objective function at al=0
//   d - [real,vec] the direction vector which the BFGS procedure
//     has chosen to perform the univariate search.
//   fmin - [real,scal] user-defined lower bound on the problem;
//     if a point is found such that the objective value is less than
//     fmin, then the procedure will terminate.
//   deltaf - [real,scal] estimated change in cost
//   alphamax - [real,scal] maximum alpha set be stability constraint.
// Output:
//   ali - [real,scal] the step-size determined to be the solution
//     to the univariate problem.
//   fal, fp0, fp1  - [real,scal] the cost function values at the point
//     (x+al*d); this is included to help keep the number of computations
//     to a minimum.
//   dfdpal,dfp0,dfp1 - [real,vec]; gradients at the min point.
//   flag - [int,scal] denotes the condition with which INACCURATE has
//     completed the line search.
//     0 - solution found;   1 - solution with function value less than fmin
//     2 - reached end of search interval, reduction without finding min
//

// The Armijo/Goldstein parameters, described in more detail in
// Fletcher, PRACTICAL METHODS OF OPTIMIZATION, Wiley, 1987.
Rho=0.01;
Sigma=0.7;
Tau1=10;
Tau2=0.1;
Tau3=0.5;

fpz=dfdpz'*d;            // slope when alpha=0, ie directional slope
alpr=0;                  // alpha(i-1) the previous alpha
falpr=fz;                // function value at previous point
fpalpr=fpz;              // slope at previous point
bounds=[1;-10*deltaf/fpz;alphamax]; // used to find inital alpha
al=min(bounds);          // initial alpha
albound=min((fmin-fz)/(Rho*fpz),alphamax); // search restricted to [0,albound]

// The following algorithm is not documented in great detail as it follows
// almost directly from Fletcher, PRACTICAL METHODS OF OPTIMIZATION, Wiley,
// 1987 pp.34,35.  The only part of the code  not in the book is
// the minimization of the cubic interpolation of fal, fpal, falp
// and fpalp used to generate the next alpha in both the bracketing
// and the sectioning phases

// The bracketing phase
while 1>0 do ...
 [dfdpal,dfp0,dfp1,fal,fp0,fp1]=COST(x+al*d,lambda); ...
 fpal=dfdpal'*d;...
 al,fal,fpal, alpr, falpr, fpalpr,...
 if fal<=fmin then flag=1; ali=al; retf; end;...
 if fal>fz+al*Rho*fpz,a=alpr;b=al;fa=falpr;fb=fal;fpa=fpalpr;fpb=fpal;...
  exit;end;...
 if fal>falpr, a=alpr;b=al;fa=falpr;fb=fal;fpa=fpalpr;fpb=fpal;exit;end;...
 if abs(fpal)<=-Sigma*fpz, flag=0;ali=al;retf;end;...
 if fpal>=0, b=alpr;a=al;fb=falpr;fa=fal;fpb=fpalpr;fpa=fpal;exit;end;...
 if abs(al-albound)<0.001*albound, flag=2;ali=al;retf;end;...
 if albound<=2*al-alpr then m=(al+albound)/2; n=albound;...
  else m=2*al-alpr; n=min([albound,al+Tau1*(al-alpr)]);end;...
 dal=al-alpr; dfal=(fal-falpr)/dal; dfpal=(fpal-fpalpr)/dal;...
 c4=(dfpal-2*(dfal-fpalpr)/dal)/dal; c3=0.5*dfpal-1.5*dal*c4;...
 if c4==0, if c3==0, alnew=-(abs(fpalpr)/fpalpr)*albound;...
  else alnew=alpr-fpalpr/(2*c3);end;...
  elseif c3*c3-3*c4*fpalpr<=0, alnew=-(abs(fpalpr)/fpalpr)*albound;...
  else alnew=al+(sqrt(c3*c3-3*c4*fpalpr)-c3)/(3*c4); end;...
 alpr=al; falpr=fal; fpalpr=fpal;...
 if alnew<=m, alnew=m;elseif alnew>=n, alnew=n;end,...
 al=alnew;...
end;
//  The steps above select a new alpha in the interval
//  [al+(al-alpr),al+Tau1*(al-alpr)].  alpha is chosen to minimize
//  a cubic polynomial which interpolates the values of f(al), f'(al),
//  f(alpr) and f'(alpr).  Note that the new point is chosen by moving far out
//  to the right in an effort to bracket an interval of acceptable points.
//
//  Similarly, in the sectioning phase, construct a cubic interpolant
//  using the values of f(A), f(B), f'(A) and f'(B) and find a minimum point
//  which (hopefully) lies in the interval
//  [(1-Tau2)*A+ Tau2*B,Tau3*A+(1-Tau3)*B], 0<Tau2<Tau3<1.  The new
//  point is chosen in this interval (which is a subinterval of its
//  predecessor).  Eventually an acceptable point will be found.

// The sectioning phase
while 1>0 do ...
 a,fa,fpa, b,fb,fpb,...
 dal=b-a; dfal=(fb-fa)/dal; dfpal=(fpb-fpa)/dal;...
 c4=(dfpal-2*(dfal-fpa)/dal)/dal; c3=0.5*dfpal-1.5*dal*c4;...
 if c4==0, if c3==0, alnew=b; else alnew=a-fpa/(2*c3);end;...
   elseif c3*c3-3*c4*fpa<=0, alnew=b;...
   else alnew=a+(sqrt(c3*c3-3*c4*fpa)-c3)/(3*c4);end;...
 m=a+tau2*dal;  n=b-tau3*dal;...
 if m<n, if alnew<=m, alnew=m; elseif alnew>=n, alnew=n; end,...
  else  if alnew<=n, alnew=n; elseif alnew>=m, alnew=m; end,end;...
 [dfdpal,dfp0,dfp1,fal,fp0,fp1]=COST(x+alnew*d,lambda);...
 fpal=dfdpal'*d;...
 if fal<=fmin ,flag=1; ali=alnew;  retf; end;...
 if fal>fz+Rho*alnew*fpz, b=alnew;fb=fal;fpb=fpal;...
  elseif fal>fa,b=alnew;fb=fal;fpb=fpal;...
  else if abs(fpal)<=-Sigma*fpz,flag=0;ali=alnew;retf; end;...
   aold=a; faold=fa;  fpaold=fpa; a=alnew; fa=fal; fpa=fpal;...
   if 0<=(b-aold)*fpal, b=aold; fb=faold; fpb=fpaold; end;...
 end;...
end;

retf
//
//  The function INACCURATE in the file 'linesrch.mtx' performs Fletcher's
//  inaccurate line search as a part of the unconstrained optimization
//  procedure, PARTITION.  It makes calls to the user-defined function (UDF)
//  COST in 'cost0.mtx' which produces both the function value and its
//  gradient.
//
//  INACCURATE will solve the univariate minimization problem required
//  at each line search in the overall optimization process, that is,
//
//                          min f(x+al*d)
//
//  This is accomplished by use of a two-stage (bracketing/sectioning)
```

```
//  procedure which employs cubic approximation that will seek
//  only to find a point which satisfies the Armijo-Goldstein conditions.
//  This approach is useful because the BFGS method of
//  determining a search direction has proven to be robust enough to
//  be effective even with an inaccurate line search.
//
//  This line search makes use of many parameters that define various
//  aspects of the search computations.  These parameters have been set
//  to values that have proven to be adequate for this procedure.  Of
//  course, these may be changed to suit the user's needs.  For a more
//  thorough description of how each parameter is used in this procedure,
//  see Fletcher, PRACTICAL METHODS OF OPTIMIZATION, Wiley, 1987.
//
//  Parameters: The Armijo/Goldstein parameters should satisfy:
//
//              0 < Rho < 1/2 --- smaller is easier to satisfy; we use 0.01
//              Rho < Sigma < 1 --- Larger is easier to satisfy; we use .7
//
//              The next is used in the bracketing stage to find (if
//              necessary) larger and larger intervals which might contain an
//              acceptable, i.e. Armijo/Goldstein, interval:
//
//              Tau1 > 1 --- we use 10, the larger, the more effort might be
//                           used in the sectioning phase.
//
//             The last two parameters are used to reduce the acceptable
//             interval until an acceptable point is found:
//
//              0< Tau2 < Sigma is advised --- we use 0.1
//              Tau2 < Tau3 <= 1/2 --- we use .5 to get the greatest
//                                 reduction of size.
//
//   Notes on the use of the output variable 'flag' in this function:
//     Flag is used by INACCURATE.DAT to characterize the condition of
//     the line search when it terminates.  The values can signify:
//       flag=0 - the solution to the line search has been found;
//                nothing special to note about the solution.
//       flag=1 - a point has been found that gives an objective value
//                less than fmin.
//       flag=2 - the bracketing step has converged to the right hand
//                endpoint of the interval without satisfying the A-G
//                conditions.  It should be noted however that the new point
//                will have a reduced function value (although, perhaps
//                insufficient to trigger the A-G conditions).

//  Created:  01/30/90
//  Programmer:  Steven Ims
//  Rewritten by Phil Schmidt and students Nader Kamrani and Brian Holawecky at
//  The University of Akron with the support of NASA Lewis Research Center
//  under grant NAG-3-1146.
```

```
// [sm]=modl(s,ns)
//
// The function MODL in file MODL.MTX puts a system matrix into a modified
// modal form reducing the number of parameters in the optimization.
// The inputs are a system matrix and its order.  The system is put into the
// form where A has 2x2 companion matrix blocks whose first rows are [0 1] and
// whose second rows are [a b].  The transformation matrix is normalized by
// requiring that all nonzero entries in the first column of the B matrix
// remain fixed (in case of zero entries in B1, the entry is fixed at 1e-9).
// The C, and D are full.  This should work except in the special case of
// repeated real roots.
// Input:
//   s - the system matrix to be put in modal form
//   ns - the order of the input matrix
// Output:
//   sm - the system matrix, now in modified modal form, still has order ns
//

// Convert the input system matrix S into the internal Matrixx modal form
// where the A submatrix is of the form that the real eigenvalues
// are on the diagonal and the complex conjugate eigenvalues (a + bi)
// and (a - bi) are stored in 2x2 matrices of the form [a b] on the first
// row and [-b a] on the second row.
[sm,t]=modal(s,ns);

// Split the modal form sm into its components
[am,bm,cm,dm]=split(sm,ns);

// Create a permutation matrix P which collects all the real distinct
// eigenvalues at the top of am and sends all the 2x2 matrices representing
// the complex eigenvalues to the bottom of am.
i=1; j=0; p=eye(ns);
[n,o]=size(cm);
while i < ns,...
 if am(i,i+1)=0,i=i+1;...
 else,...
  m=p([i:i+1],:);...
  if i>j+1, for k=i:-1:j+2, p(k+1,:)=p(k-1,:); end; end;...
  p([j+1:j+2],:)=m;...
  j=j+2; i=i+1;...
 end;...
end;

// Rearrange the real eigenvalues so that the ones that are close
// in value are separated.
nr=ns-j;
if nr > 2, q=mod(nr,2); nrhalf=(nr-q)/2; tpnt=j+2; mpnt=j+nrhalf+1;...
 for i=1:(nrhalf-1), temp=p(mpnt,:);...
   p([(tpnt+1):mpnt],:)=p([tpnt:(mpnt-1)],:);...
   p(tpnt,:)=temp; tpnt=tpnt+2; mpnt=mpnt+1;...
 end;...
end;

// Since S=[A,B; C,D] is the form of a system matrix, multiplying
// [P, I] * S * [inv(P); I] will have the following effect on A,B,C,D
a=p*am*inv(p);
b=p*bm;
c=cm*inv(p);

// If two real eigenvalues are still very close in value, shift one slightly
// more negative
for i=1:nr-1, ...
  if a(i,i)=a(i+1,i+1), a(i,i)=a(i,i)-10e-9; end;
end;

// Construct a transformation matrix T such that
// T * a * inv(T) has the effect of taking 2x2 blocks of
// real eigenvalues (or of complex conjugate pairs) and
// returning the desired structure [0 1; a b] which have the same
// eigenvalues as the 2x2 blocks.
t=0*eye(ns);
q=mod(ns,2);
for i=1:(ns-q)/2,...
  t(2*i-1,2*i-1)=1; t(2*i-1,2*i)=1;...
  t(2*i,2*i-1)=a(2*i-1,2*i-1)-a(2*i-1,2*i);...
  t(2*i,2*i)=a(2*i,2*i)-a(2*i,2*i-1);...
end;

// The variable q keeps track of an odd size matrix, i.e. an odd entry which
// does not fit in the 2x2 blocks
if q=1, t(ns,ns)=1; end;

// [T I] * system matrix * [inv(T); I] will generate the following
tinv=inv(t);
aa=t*a*tinv; bb=t*b; cc=c*tinv;

// Set the first entry in the modal blocks to zero; [0,1;a,b]
for i=1:(ns-q)/2, aa(2*i-1,2*i-1)=0; end;

// Construct the system matrix sm which is now in the modified modal form
sm=[aa,bb;cc,dm];

retf

// Created by Phil Schmidt and students Nader Kamrani and Brian Holawecky at
// The University of Akron with the support of NASA Lewis Research Center
// under grant NAG-3-1146.
```

```
// [ska,ske]=mat(p)
//
// The function MAT in file PARMAT.MTX generates
// the partitioned system matrices ska and ske from
// the parameter vector p.  Note that p is: Aa,Ae,Ba,Ca,Deaa,Deaya,Be,Ce,Deea.
// The other submatrices are all constants and are loaded from const.dat
// Input:
//   p - the vector of parameters
//   const.dat - datafile of constants
// Output:
//   ska - the airframe controller
//   ske - the engine controller
//

// load the default data from const.dat
load 'const.dat' nka ma lya nke bal ka pea me ke lye be1 ...
  daa daya dee deye ska0 ske0 aore fixd

// Set the counter, first to 0 then to nka etc. as new blocks are built from p
dum = 0;

// If airframe is not fixed, then create a blank aa (A submatrix for ska)
// and copy the requisite entries from the parameter vector p.  Recall  the
// modal from of the system matrices (refer to modl.mtx) reduces the
// A submatrices to 2x2 companion blocks of the form [0,1; a,b], so only
// the a and b need to be read
if aore<>1, aa=0*eye(nka); q=mod(nka,2); ...
 for i=1:(nka-q)/2,...
  aa(2*i-1,2*i)=1; aa(2*i,2*i-1)=p(2*i-1,1); ...
  aa(2*i,2*i)=p(2*i,1); end;...
 if q=1, aa(nka,nka)=p(nka,1); end; ...
 matot=ma+lya; katot=ka+pea; dum=nka;...
end,

// If engine is not fixed, then create a blank ae and copy requisite
// entries from the p vector, with the same provisos as for aa above
if aore<>2, ae=0*eye(nke); q=mod(nke,2); ...
 for i=1:(nke-q)/2,...
  ae(2*i-1,2*i)=1; ae(2*i,2*i-1)=p(dum+2*i-1,1); ...
  ae(2*i,2*i)=p(dum+2*i,1);end;...
 if q=1, ae(nke,nke)=p(dum+nke,1);  end; ...
 metot=me+pea+lye; dum=dum+nke;...
end,

// If airframe is not fixed, generate the ca,deac,da entries from p.
// Note that the first column of the ba entry is fixed as required by
// the modal form, so load it from ba1 (saved in const.dat)
// The rest of ba,ca,deac are loaded from p. If Ds are not fixed, then
// their values are read from p, otherwise from constants
// (and daya depending on whether feedback lya exits).
if aore<>1, ba(:,1)=ba1;..
 for i=2:matot, ba(:,i)=p([dum+1:dum+nka],1); dum=dum+nka; end;..
 for i=1:nka, ca(:,i)=p([dum+1:dum+katot],1); dum=dum+katot; end;..
 if (fixd-1)*(fixd-2) <> 0,...
   for i=1:matot, daaya(:,i)=p([dum+1:dum+ka],1); dum=dum+ka; end;..
 else if lya=0, daaya=daa; else daaya=[daa, daya]; end;...
 end;...
 for i=1:matot, deac(:,i)=p([dum+1:dum+pea],1); dum=dum+pea; end;..
 da=[daaya;deac];...
end,

// If the engine controller is not fixed, load the first column of be
// from the be1 entry stored in const.dat.  Then load rest of be,ce,deea
// from p.  Construct de from deea, dee (and deye depending on whether
// there is any feedback lye).
if aore<>2, be(:,1)=be1;..
 for i=2:metot, be(:,i)=p([dum+1:dum+nke],1); dum=dum+nke; end;..
 for i=1:nke, ce(:,i)=p([dum+1:dum+ke],1); dum=dum+ke; end;..
 if (fixd-2)*(fixd-3)=0, deea=0*ones(ke,pea);...
   else for i=1:pea, deea(:,i)=p([dum+1:dum+ke],1); dum=dum+ke; end;...
 end; ...
 if (fixd-1)*(fixd-2) <> 0,meplye=me+lye;...
   for i=1:meplye, deeye(:,i)=p([dum+1:dum+ke],1); dum=dum+ke; end;..
 else if lye=0, deeye=dee; else deeye=[dee, deye]; end;...
 end;...
 de=[deea, deeye];...
end,

// If the airframe controller is not fixed, then construct ska from aa,ba
// ca and da.  If there is no feedback defined lya=0, then extend
// ska by one column of zeros to accomodate the dummy feedback that will
// be used.
// If the airframe controller is fixed, then use ska=ska0
if aore<>1,  ska=[aa,ba;ca,da];...
 if lya=0, [rska,cska]=size(ska); ska=[ska,0*ones(rska,1)];end,...
else ska=ska0;...
end,

// If the engine controller is not fixed, then construct ske.
// If there is no feedback defined lye=0, then add the zero column
// to accomodate the dummy feedback.
// If the engine controller is fixed, then let ske=ske0 the fixed
// engine controller.
if aore<>2, ske=[ae,be;ce,de];...
 if lye=0,  [rske,cske]=size(ske); ske=[ske,0*ones(rske,1)]; end,...
else ske=ske0; end,

retf

// This program takes the parameter vector P, which is a column vector, and
// uses the dimensions of the subcontrollers stored in 'const.dat' to
// reconstruct the partitioned system matrices ska and ske.  It allows for the
// possibility that one of the subcontrollers is fixed and loads its initial
// value.  Note that the constant submatrices Ba1,Be1, Daa, Daya, Deac, Dee and
// Deye are properly loaded.

// Created by Phil Schmidt and students Nader Kamrani and Brian Holawecky at
// The University of Akron with the support of NASA Lewis Research Center
// under grant NAG-3-1146.
```

```
// p_i=partition(stop)
//
// The function PARTITION in file PARTITIO.MTX is the
// outer level loop in the optimization process.
// It implements the Broyden-Fletcher-Goldfarb-Shanno method of finding
// a search direction for minimization, then calls function INACCURATE
// to generate a new point by minimizing in that direction.  This
// process is then repeated until the convergence conditions (as
// checked by function CONVERGE) are met or some other stopping criteria
// are met (number of iterations exceeds maximum or function value is
// less than fmin).
// Input:
//   stop - the vector of stopping conditions
//   const.dat - the datafile of constants
//   inter.dat - the datafile containing intermediate results
// Output:
//   p_i - the ith point along the optimization process
//   inter.dat - the datafile where intermediate results are stored
//

load 'const.dat' omega nka nke atot stabil
load 'inter.dat' p_i gradi grad0 grad1 fh jh0 jh1 hi lambda fx;
[row,col]=size(p_i); [cnt,tmp]=size(fh);

// Prestore the messages to save the length of loops, in order that the
// Matrixx line limit be avoided
ls='LINESEARCH TERMINATED BY HITTING BOUND';
not='NOTICE! STABILITY BOUND.';
a_ph='A phat has been found that gives an objective value < fmin.';
ex='exceeded maximum number of iterations';
plpar='STRIP YLAB/f(P)|f0(P)|f1(P)/';

// Read stopping conditions  from the vector stop
epsl =stop(1)*ones(p_i);
del =stop(2);
eta = stop(3);
maxcnt =stop(4);
fmin =stop(5);

dfx=diag(ones(row,1)-[fx;0*ones((row-atot),1)]);
delf=max(0.9*abs(fh(cnt)),10**(-6));
cvgfact=1;
cvflg=0;
// While the convergence/stopping conditions have not been met
while cvflg<0.5 do...
 cnt,fh(cnt), ...
 if cnt>1, ...
  if cnt>20, ITER=[cnt-20:cnt]'; ...
  else ITER=[1:cnt]';...
  end..
  Y=[fh(ITER),jh0(ITER),jh1(ITER)];PLOT(ITER,Y,plpar);...
 end,...
 di=-dfx*hi*dfx*gradi;...// search direction is -Approx Inv Hessian*gradient
 alphamax=1e16;...// check on stability constraints
 for I=1:atot,...
  if fx(I)=0,...
   if di(I)>0,...
    if (stabil-p_i(I))/di(I)<alphamax,...
     alphamax=(stabil-p_i(I))/di(I);...
    end,...
   end,...
  else di(I)=0; end,...
 end,...
 [ali,fp_i1,j0,j1,gradi1,grad0,grad1,flag]=INACCURATE(p_i,fh(cnt),gradi,...
 di,fmin,delf,lambda,alphamax);...// calls the linesearch
if flag=2, display(ls); end;...
if ali=alphamax, cvgfact=0; display(not); end,...
fh(cnt+1)=fp_i1;...
p_i1=p_i+ali*di; ...
jh0(cnt+1)=j0; ...
jh1(cnt+1)=j1;...
cvflg=CONVERGE(p_i1,fh(cnt+1),p_i,fh(cnt),gradi1,cvgfact*epsl,...
  cvgfact*del,eta,fx);...// checks convergence
cvgfact=1;...
psi=ali*di;...
Qi=dfx*(gradi1-gradi);...
psiQi=psi'*Qi;...
pQmat=(eye(row)-psi*Qi'/psiQi);...
hi=pQmat*hi*(pQmat')+((psi*psi')/psiQi);...// updates Inv Hessian Approx
for I=1:atot, ...
 if fx(I)=0, if p_i1(I)>=1.001*stabil, fx(I)=1;...
  p_i1(I)=stabil;dfx(I,I)=0;end...
 elseif gradi1(I)>0,fx(I)=0; for J=1:row,hi(I,J)=0;hi(J,I)=0;end,...
  hi(I,I)=1;dfx(I,I)=1;...
 end,end ...// updates stability constraints
p_i=p_i1;...
gradi=gradi1;...
delf=max(fh(cnt)-fh(cnt+1),10**(-6));...
cnt=cnt+1;...
if mod(cnt,20)=0, ...
 save 'inter.dat' p_i fh jh0 jh1 gradi grad0 grad1 lambda fx hi;end...
if cnt=maxcnt,disp(ex);...
 save 'inter.dat' p_i fh jh0 jh1 gradi grad0 grad1 lambda fx hi;...
 retf;end,...
if flag=1,disp(a_ph); ...
 save 'inter.dat' p_i fh jh0 jh1 gradi grad0 grad1 lambda fx hi;...
 retf;end;...
end;
save 'inter.dat' cnt p_i fh jh0 jh1 gradi grad0 grad1 lambda fx hi;
disp('Convergence of f-value and parameters has occurred');
if cvflg=2, disp('all partials are also less than'); eta, end;
retf

//  PARTITION is the top-level function in the optimization routine.
//  It makes calls to the user-defined functions INACCURATE and  CONVERGE
//  in the files 'linesrch.mtx' and  'converge.mtx' resp.
//  It also uses the MATRIXX PLOT function to show intermediate results.
//
//  PARTITIO.MTX is an implementation of the Broyden-Fletcher-Goldfarb-Shanno
//  method of determining a search direction, with Fletcher's inaccurate
//  line search being used to locate the the next point in UDF INACCURATE.
//
//  PARTITION.MTX will solve hierarchical partitioning problems.
//
//  Inputs:
//           stop = [real,vec] column vector of stopping conditions
//           stop = [epsl del eta maxcnt fmin]
//
//   Parameters being defined in 'param' vector: any defaults were assigned in
//   START
//
//      epsl     (default=1e-9) - column vector with element-wise convergence
//                                  limits for elements of |p_i1 - p_i|;
//      del      (default=1e-9) - convergence of objective values,
//                                  |f(p_i1) - f(p_i)|;
//      eta      (default=1e-9) - convergence of gradients,
//                                  |grad f(p_i1)| < eta;
//      maxcnt   (default=100)  - maximum number of iterations of procedure
```

```
//                                  allowed in searching for a solution.
//      fmin     (default=0)     - lower bound on the objective function.
//                                  Any point, x, found with an objective
//                                  value less than fmin will be considered
//                                  a solution. Since we minimize a norm, the
//                                  default is zero.
//
//   Notes on the use of 'flag' in this function:
//      Flag is used by the INNACURATE functions to
//      characterize the objective function and the current point, x.
//      It can take on values as follows:
//      (1) flag=0 - a point has been found which satisfies the Armijo-
//                   Goldstein conditions (see 'linesrch.mtx'); the precedure
//                   continues.
//      (2) flag=1 - a point has been found that gives an objective value
//                   less than fmin. Procedure terminates.
//      (3) flag=2 - a point has been returned by INACCURATE which doesn't
//                   satisfy the A-G condition, but which corresponds to the
//                   maximum allowable step in the search direction.  This can
//                   happen because the steps are constrained to maintain
//                   subcontroller  stability.  Currently, the code continues
//                   as though the A-G conditions were satisfied.  If problems
//                   such as looping are encountered, the user might try to
//                   reset HI to the identitiy at this point.
//
//  The program terminates properly under one of the following conditions:
//
//        (1) flag=1    - the objective value is less than fmin.
//        (2) cnt=mxcnt - the maximum number of iterations has occurred
//        (3) cvflg=1   - the max deviation in parameters is less than epsl and
//                        the max deviation in COST is less than delta
//        (4) cvflg=2   - same conditions as cvflag=1 and max deviation in
//                        partials is less than eta.
//
//  On termination (of any type) the following can be found in the file
//  'inter.dat':
//
//        p_1   - last parameter vector (can use procedure MAT in 'parmat0.mtx'
//                to generate the corresponding SKA and SKE)
//        fh    - complete total cost history (all iterations)
//      jh0, jh1 - complete cost histories of fPerf and fTrack costs
//        grad1 - gradient of total cost at last parameter point.
// grad0, grad1 - gradients of fPerf and fTrack at last point
//        hi    - last inverse Hessian updated according to values of p_1, etc.
//       lambda - lambda vector
//        fx    - vector indicating which entries in Aa and Ae are fixed at
//                stability bounds

//  Created:  01/23/90 as BFGS.DAT
//  Programmer:  Steven Ims
//  Revisions by Phil Schmidt and students Nader Kamrani and Brian Holawecky at
//  The University of Akron with the support of NASA Lewis Research Center
//  under grant NAG-3-1146.
```

```
// p=longcol(ska,ske)
//
// The function LONGCOL in file PARVEC.MTX
// creates the long column vector of parameters p from the partitioned
// system matrices ska and ske
// Input:
//   ska - the airframe controller state-space matrix
//   ske - the engine controller state-space matrix
//   const.dat - datafile of constants
// Output:
//   p - the long column vector of the parameters
//

load 'const.dat' nka nke me pea lye ma lya ka aore fixd

[aa,ba,ca,da]=split(ska,nka);   // split ska and ske into their
[ae,be,ce,de]=split(ske,nke);   // component [A,B;C,D]

metot=me+pea+lye;               // define sizes of the parameters
matot=ma+lya;

p=[0];                          // initialize p

// If airframe controller is not fixed then copy entries that need to be
// optimized from aa.  Note that because aa has the modified modal from
// consisting of 2x2 companion matrices [0,1 ; a,b], only the a,b need to
// be included into the parameter vector.  The last block of aa may contain
// only [a,b] and needs to be included if a is odd (q=1)
if aore<>1,q=mod(nka,2);...
 for i=1:(nka-q)/2, p=[p;aa(2*i,2*i-1); aa(2*i,2*i)]; end;...
 if q=1, p=[p;aa(nka,nka)]; end;..
end,

// If engine controller is not fixed, put the required elements of
// ae into p (same modified modal form as aa)
if aore<>2,q=mod(nke,2);...
  for i=1:(nke-q)/2,p=[p;ae(2*i,2*i-1);ae(2*i,2*i)];end;...
 if q=1, p=[p;ae(nke,nke)]; end;...
end,

// If airframe controller is not fixed, then copy ba (note first column
// of ba is fixed and is not copied), ca and da into p
if aore<>1,...
 for i=2:matot, p=[p;ba(:,i)]; end;..
 for i=1:nka, p=[p;ca(:,i)]; end;..
 if (fixd-1)*(fixd-2) <> 0, for i=1:matot, p=[p;da([1:ka],i)]; end; end;...
 for i=1:matot, p=[p;da([ka+1:ka+pea],i)]; end;...
end ,

// If the engine controller is not fixed copy be (first col fixed), ce and
// de into p
if aore<>2,...
 for i=2:metot, p=[p;be(:,i)]; end;...
 for i=1:nke, p=[p;ce(:,i)]; end;...
 if (fixd-2)*(fixd-3) <> 0, for i=1:pea, p=[p;de(:,i)]; end; end;...
 if (fixd-1)*(fixd-2) <> 0, for i=pea+1:metot, p=[p;de(:,i)]; end; end;...
end,

// Get rid of the 0 which was the 1st entry in p (used to initialize p)
[x y]=size(p);
p=p([2:x],1);

retf
```

```
// This program accepts the partitioned system matrices of subcontrollers
// and generates the long column vector of the parameters. The order with
// which this procedure builds p is: Aa,Ae,Ba,Ca,Deaa,Deaya,Be,Ce,Deea.  Note
// that it skips matrices if the subcontroller is fixed.  It also ignores the
// submatrices which are constants --- Bal, Bel, Daa, Daya, Dee, and Deye.


// Created by Phil Schmidt and students Nader Kamrani and Brian Holawecky at
// The University of Akron with the support of NASA Lewis Research Center
// under grant NAG-3-1146.
```

```
// [ska_opt,ske_opt]=restart(lambdanew,stop,newh)
//
// The function RESTART in file RESTART.MTX is used to
// restart the optimization process after it has been intentionally
// halted or has stopped due to the stopping criteria.
// Some parameters may be altered if desired.
// Input:
//   lambdanew - a new value for lambda, the weighting of the tracking
//     cost as part of the total cost function
//   stop - (optional) the vector of stopping conditions
//   newh - (optional) if any parameter whatsoever is passed, this sets
//     the inverse Hessian approximation to an identity matrix
//   const.dat - the datafile of constants
//   inter.dat - the intermediate results at the last checkpoint or the
//     final results if convergence/stopping criteria were met.
// Output:
//   ska_opt - the optimized airframe controller
//   ske_opt - the optimized engine controller

resize('sstack',1000000); // load the relevant data
load 'const.dat' nka nke pea lya lye;   atot=nka+nke;
load 'inter.dat' p_i fh jh0 jh1 gradi grad0 grad1 hi fx lambda;

short e;                // Specify the format for displaying matrices

// Define UDF INACCURATE which implements Fletchers inaccurate line search
define 'linesrch.mtx';

// Define UDF CONVERGE which checks if convergence conditions are met
define 'converge.mtx';

// Define UDF PARTITION which is the outermost loop and implements the
// Broyden-Fletcher-Goldfarb-Shanno technique for finding a search direction
// for optimization
define 'partitio.mtx';

// Define UDF Z which returns a zero matrix of the desired size
define 'zero.mtx'

// Define the UDF MAT which generates the SKA and SKE subcontrollers
// from the parameter vector p
define 'parmat.mtx'

// Define UDF LONGCOL which generates the parameter vector from the
// SKA and SKE groupings of the parameters
define 'parvec.mtx'

// Define UDF COST which evaluates the performance cost, tracking cost
// and their sum, as well as the respective gradients for a specified
//  parameter vector
define 'cost.mtx'

// if stop does not exist, load it from par.dat
if exist('stop')=0, load 'par.dat' stop

// Verify that the newlamda provided is a row vector or a scalar and is
// consistant with pea
[rlambda,clambda]=size(lambdanew);
if rlambda > 1, disp('ERROR: lambda must be a row vector or a scalar!'), retf;
if clambda=1, lambdanew=lambdanew*ones(1,pea); ...
    elseif  clambda <> pea,...
    display('ERROR: lambda must contain pea entries!'); retf;
changelam=norm(lambda-lambdanew);
lambda=lambdanew;

// If the difference between old lambda and new lambda is large enough,
// >1e-10, then call COST to initialize costs and gradients, otherwise
// just use old data
if abs(changelam)>1e-10, ...
  [gradi,grad0,grad1,fh,jh0,jh1]=COST(p_i,lambda);...
else ...
  [cnt,tmp]=size(fh); fh(cnt)=jh0(cnt)+jh1(cnt); gradi=grad0+grad1;...
end,

// If newh is defined, then set the inverse Hessian to an identity matrix,
// If any of the A parameters are at the bounds (noted in fx)
// then set that element of hi to  be zero (no further decrease in that
// direction/parameter)
if 1=exist('newh'), [row,col]=size(p_i); hi=eye(row); ...
  for i=1:atot, if fx(i)=1, hi(i,i)=0; end,end,...
end;

// save the possibly new stopping vector and save intermediate data
save 'par.dat' stop;
save 'inter.dat' p_i fh jh0 jh1 gradi grad0 grad1 hi fx lambda;

// call the PARTITION routine which eventually returns the final
// optimized vector
p_opt=PARTITION(stop);

// Generate the optimized ska and ske as the final output of the program
[ska_opt,ske_opt]=MAT(p_opt);
load 'inter.dat' p_i fh jh0 jh1 gradi grad0 grad1 hi fx lambda;
// If a dummy feedback for ya or ye was added, then remove it
if lya=0, [r,c]=size(ska_opt); ska_opt=ska_opt(:,1:c-1); end;
if lye=0, [r,c]=size(ske_opt); ske_opt=ske_opt(:,1:c-1); end;

save 'inter.dat' p_i fh jh0 jh1 gradi grad0 grad1 hi fx lambda ska_opt ske_opt;

// INTER.DAT now contains the final results including the optimal
// subcontrollers.

retf

// Created by Phil Schmidt and students Nader Kamrani and Brian Holawecky at
// The University of Akron with the support of NASA Lewis Research Center
// under grant NAG-3-1146.
```

```
// [ska_opt, ske_opt]=start(lambda,stop)
//
// The function START in file START.MTX initiates the
// controller partitioning optimization code.
// It sets up initial data and precalculates some required information,
// calls COST to initialize the costs and gradients and
// calls PARTITION to do the optimization.
// Input:
//   lambda - a scalar or pea x 1 vector of weightings for determining the
//     relative contribution of fl(p), the tracking cost, to the total cost.
//   stop - (optional) the vector of stopping conditions
//     stop=[eps;delta;eta;iter;fmin] where
//     default: stop=[1e-7;1e-7;1e-7;1e2;1e-2];
//     eps = tolerance for max change in parameters giving convergence
//     delta = tolerance for max change in cost giving convergence
//     eta = tolerance for max change in norm of gradient giving convergence
//     iter = number of iterations to run; defaults to 100
//     fmin = minimum tolerance for the total cost; a cost below this
//       is assumed to be a minimum.
//   INIT.DAT - the data file containing the initial information
// Output:
//   ska_opt - the optimized airframe subcontroller
//   ske_opt - the optimized engine subcontoller
//

resize('sstack',1000000);       // System stack size is increased
load 'init.dat';                // Initial data is loaded

// Define UDF MODL which puts a system matrix into a modified modal form
define 'modl.mtx';

// Define UDF INACCURATE which implements Fletchers inaccurate line search
define 'linesrch.mtx';

// Define UDF CONVERGE which checks if convergence conditions are met
define 'converge.mtx';

// Define UDF PARTITION which is the outermost loop and implements the
// Broyden-Fletcher-Goldfarb-Shanno technique for finding a search direction
// for optimization
define 'partitio.mtx';

// Define UDF RESTART which can restart the program using checkpointed
// information
define 'restart.mtx';

// Define UDF Z which returns a zero matrix of the desired size
define 'zero.mtx'

// Define the UDF MAT which generates the SKA and SKE collections of the
// parameters from the parameter vector
define 'parmat.mtx'

// Define UDF LONGCOL which generates the parameter vector from the
// SKA and SKE system controllers.
Define 'parvec.mtx'

// Define UDF COST which evaluates the performance cost, tracking cost
// and their sum, as well as the respective gradients for a specified
//  parameter vector
define 'cost.mtx'

short e;                // Specify the format for displaying matrices
ska_opt=0; ske_opt=0;   // Initializing answers in case of an abrupt error/end

if exist('sp')=0, if exist('spaug')=1, sp=spaug;
if exist('np')=0, if exist('nspaug')=1, np=nspaug;

// Checking the input data to see if it is consistant.
if exist('s_ka')=0, display('ERROR: s_ka is missing from init.dat'); retf ;
if exist('ns_ka')=0, display('ERROR: ns_ka is missing from init.dat'); retf;
if exist('s_ke')=0, display('ERROR: s_ke is missing from init.dat'); retf;
if exist('ns_ke')=0, display('ERROR: ns_ke is missing from init.dat'); retf;
if exist('sc')=0, display('ERROR: sc is missing from init.dat');retf;
if exist('nsc')=0, display('ERROR: nsc is missing from init.dat');retf;
if exist('sp')=0, display('ERROR: sp is missing from init.dat');retf;
if exist('np')=0, display('ERROR: np is missing from init.dat');retf;
if exist('frq')=0, display('ERROR: frq is missing from init.dat');retf;
if exist('pea')=0, display('ERROR: pea is missing from init.dat');retf;
if exist('la')=0, lya=0; else lya=la;
if exist('le')=0, lye=0; else lye=le;
[r,c]=size(s_ka); ka_tmp = r-ns_ka-pea; ma_tmp = c-ns_ka-lya;
[r,c]=size(s_ke); ke_tmp = r-ns_ke; me_tmp = c-ns_ke-pea-lye;
[r,c]=size(sp);
if r<>np+ma_tmp+me_tmp+pea+lya+lye, ...
 display('ERROR: number of rows in sp not consistant with other data'); ...
 retf; end;
if c<>np+ka_tmp+ke_tmp, ...
display('ERROR: number of columns in sp not consistant with other data');...
 retf; end;
[r,c]=size(sc);
if r<>nsc+ka_tmp+ke_tmp, ...
 display('ERROR: number of rows in sc is not consistant with other data'); ...
 retf; end;
if c<>nsc+ma_tmp+me_tmp+lya+lye, ...
 display('ERROR: number of columns in sc not consistant with other data'); ...
 retf; end;
if frq(3,1) <= 0, ...
 display('ERROR: number of observation points frq(3,1) must be positive'); ...
 retf; end;
if mod(frq(3,1),2)=0, ...
 display('ERROR: number of observation points frq(3,1) must be odd'); ...
 retf; end;
if frq(1,1) >= frq(2,1), ...
 display('ERROR: min. observation point frq(1,1) must be less than the ');...
 display('max. observation point frq(2,1)'); retf; end;

// Get row and column of the centralized controller sc
[rsc,csc]=size(sc);
// The input and output weighting matrices swi and swo of orders nwi and nwo
// resp. can be defined in init.dat.  If not defined, they are assumed to
// be identity matrices of the desired size  (depending on the size of
// the centralized controller)
if exist('swi')=0, nwi=1; swi=eye(1+csc-nsc); end
if exist('swo')=0, nwo=1; swo=eye(1+rsc-nsc); end

if exist('nwi')=0, display('ERROR: nwi is not defined in init.dat');retf;
if exist('nwo')=0, display('ERROR: nwo is not defined in init.dat');retf;
[r,c]=size(swi);
if r<>nwi+ma_tmp+me_tmp+lya+lye, ...
 display('ERROR: number of rows in swi is not consistant with other data');...
 retf; end;
if c < nwi, display('ERROR: number of columns in swi is less than nwi'); ...
 retf; end;
[r,c]=size(swo);
if c<>nwo+ka_tmp+ke_tmp, ...
 display('ERROR: number of columns in swo not consistant with other data');...
 retf; end;
```

```
if r < nwo, display('ERROR: number of rows in swo is less than nwo'); ...
 retf; end;

// Verify that the lamda provided is a row vector or a scalar and is
// consistant with pea

[rlambda,clambda]=size(lambda);
if rlambda > 1, disp('ERROR: lambda must be a row vector or a scalar!'), retf;
if clambda=1, lambda=lambda*ones(1,pea), ...
    elseif  clambda <> pea,...
    display('ERROR: lambda must contain pea entries!'); retf;

// The stopping conditions vector is given a default value if it was not
// provided (it will be saved in par.dat)
if exist('stop')=0, stop=[1e-7;1e-7;1e-7;1e2;1e-2];
save 'par.dat' stop

// Inquire if the airframe or the engine submatrices are to be held fixed
// storing the reply in variable aore
// If not, put the subcontroller matrices in modified modal form.
// Make ska and ske with orders nka and nke the variables to be used
// from now on (instead of s_ka and s_ke)
nka=ns_ka;  nke=ns_ke;
inquire aore 'ENTER 1 TO FIX AIRFRAME, 2 TO FIX ENGINE or 0 FOR NEITHER: ';
if aore=0, atot=nka+nke; ska=modl(s_ka,ns_ka); ske=modl(s_ke,ns_ke);...
  elseif aore=1, atot=nke; ska=s_ka; ske=modl(s_ke,ns_ke);...
  elseif aore=2, atot=nka; ska=modl(s_ka,ns_ka); ske=s_ke;...
  else DISPLAY('YOU MUST ENTER 0, 1 or 2!'); retf;...
end,
// Inquire if any of the D submatrices are to be fixed.  DAA, DAYA,
// DEE, and DEYE are determined by the corresponding submatrices of the
// centralized controller. Under option 1 DEEA and DEAA are determined
// by optimization.  Under option 2 DEEA is set to 0 and DEAA is determined
// optimization.  Under option 3 DEEA=0 and the remaining Ds are determined
// by optimization.  Under option 0 no Ds are fixed.
DISPLAY('ENTER 1 TO FIX ALL Ds EXCEPT DEAA & DEEA, 2 TO INCLUDE DEEA,');
inquire fixd '      3 FOR ONLY DEEA, or 0 for NONE:';
if (fixd-1)*(fixd-2)*(fixd-3)*fixd <> 0,...
  DISPLAY('YOU MUST ENTER 0, 1, 2 or 3!'); retf;end,

// Get the row and column size of the plant matrix, input weighting matrix,
// and the airframe and engine controller matrices.
[rsp,csp]=size(sp);
[rswi,cswi]=size(swi);
[rska,cska]=size(ska);
[rske,cske]=size(ske);

// If the measurements from plant to controllers la,le, are absent,
// create a dummy feedback of one (la = 1 and/or le = 1).
// lya and lye hold the actual number of ya and ye measurements.
// Increase the sizes of the system matrices to accomodate these
// feedback variables.  In the case of the dummy feedbacks being created
// (and not input via init.dat) the system matrices are adjusted with
// zero columns.
if exist('la')=0, ...
  la=1; lya=0; ...
  ska=[ska, 0*ones(rska,1)];...
  if exist('le')=0, ....
    le=1;lye=0;...
    sc=[sc, 0*ones(rsc,2)];...
    sp=[sp; 0*ones(2,csp)];...
    swi=[swi; 0*ones(2,cswi)];...
    ske=[ske, 0*ones(rske,1)];...
  else lye=le;...
    sc=[sc(:,[1:csc-le]),0*ones(rsc,1),sc(:,[csc-le+1:csc])];...
    sp=[sp([1:rsp-le],:); 0*ones(1,csp); sp([rsp-le+1:rsp],:)];...
    swi=[swi([1:rswi-le],:); 0*ones(1,cswi); swi([rswi-le+1:rswi],:)];...
  end;...
else ...
  lya=la; ...
  if exist('le')=0, ...
    le=1; lye=0; ...
    sc=[sc, 0*ones(rsc,1)];...
    sp=[sp; 0*ones(1,csp)]; ...
    ske=[ske, 0*ones(rske,1)];...
    swi=[swi; 0*ones(1,cswi)];...
  else...
     lye=le;...
  end,...
end,

// Decompose the airframe subcontroller in modified modal form
// ska=[aa,ba;ca,da].  Use da to figure out the total inputs matot
// and outputs katot for ska.  Subtract the la input from matot
// to get ma, the actual number of inputs to ska.  Subtract the number
// of pea airframe to engine commands from the katot to get ka the number
// of outputs going to the plant
[aa ba ca da]=split(ska,nka);
[katot matot]=size(da);
ma=matot-la;
ka=katot-pea;

// Decompose the engine  subcontroller in modified modal form
// ske=[ae,be;ce,de]. Use de to find ke the number of outputs
// from the engine to the plant.  Also find metot, the total number of
// inputs, and subtratk pea, the airframe to plant commands, and le, the
// engine feedback to get me, the actual number of external inputs
// to the engine subcontroller
[ae be ce de]=split(ske,nke);
[ke metot]=size(de);
me=metot-pea-le;

// If any of the entries of the first column of ba or be are <= zero (or
// almost <= 0) then set them to 1e-9 (almost zero but positive).
// This is required by the modified modal form of the system matrix.
for i=1:nka, if abs(ba(i,1))<1e-9, ba(i,1)=1e-9; end,end,  ba1=ba(:,1);
for i=1:nke, if abs(be(i,1))<1e-9, be(i,1)=1e-9; end,end,  be1=be(:,1);

// Decompose the integrated plant sp=[ap,bp;cp,dp]
[ap,bp,cp,dp]=split(sp,np);

[kptot ntmp]=size(cp);
kplnt=kptot-pea;
[ntmp mptot]=size(bp);
kpmpea=kptot-pea+1;

// Let sk=sc be the centralized  controller
sk=sc; nk=nsc;

// Decompose the centralized  controller sk=[ak,bk;ck,dk]
[ak,bk,ck,dk]=split(sk,nk);

// If the user has not entered the (optional)  matrix Nperf for
// normalizing the performance cost, it is set to a matrix of ones
// This matrix should be a frq(3,1)x1 matrix.
if exist('Nperf')=1, ...
  Nperf = max(Nperf,1e-13); [r,c]=size(Nperf);...
  if r<>frq(3,1),...
```

```
  display('ERROR: The number of performance weights is different from the ...
  number of frequency points ');retf;end;...
else, Nperf=ones(frq(3,1),1);...
end;

// If the user has not entered the (optional)  matrix Ntrack for
// normalizing the tracking cost, it is set to a matrix of ones.
// This matrix should be a frq(3,1)xpea matrix.
if exist('Ntrack')=1, ...
  Ntrack = max(Ntrack,1e-13); [r,c]=size(Ntrack);...
  if r<>frq(3,1),...
  display('ERROR: The number of tracking weights disagrees with the ...
  number of frequency points ');retf;end;...
  if c<>pea,...
  display('ERROR: The number of tracking weights disagrees with the ...
  number of interface variables ');retf;end;...
else, Ntrack=ones(frq(3,1),pea);...
end;

// Define the weights for Simpsons rule integration.  They depend on the
// number of observation points specified in frq[3] and are stored in
// the variable WEIGHT.
weight=[1];
for i=1:((frq(3,1)-3)/2),weight=[weight,4,2];end
weight=[weight,4,1];

// Define the set of frequency points at which the cost function and gradient
// are evaluated. Store as the variable OMEGA..
omega=frq(1)*(((frq(2)/frq(1))**(1/(frq(3)-1)))**[0:frq(3)-1]);

// Decompose  the plant controller, splitting up the matrix bp relative to
// its inputs and the matrix cp relative to its outputs.  For example,
// cap would be the plant to airframe piece, the first ma rows of the
// plant matrix because these rows form the input to the airframe.
[mp ntmp]=size(cp);
kp= ka+ke; kap1= ka+1;
cap= cp([1:ma],:); ceap= cp([(ma+1):(ma+pea)],:);
cep= cp((ma+pea+1):(ma+pea+me),:);
cyap=cp([(ma+me+pea+1):(ma+me+pea+la)],:);
cyep=cp([(ma+me+pea+la+1):(ma+me+pea+la+le)],:);
bpa= bp(:,[1:ka]); bpe= bp(:,[(ka+1):kp]);

// The centralized controller is decomposed in a similar way as the plant
// controller
[kc ktmp]=size(ck);
bak= bk(:,1:ma);  bek= bk(:,[(ma+1):(me+ma)]);
byak= bk(:,[(ma+me+1):(ma+me+la)]); byek=bk(:,[(ma+me+la+1):(ma+me+la+le)]);
cak= ck(1:ka,:);  cek= ck(kap1:kc,:);
dak= dk([1:ka],[1:ma]); daek=dk([1:ka],[ma+1:ma+me]);
dayek= dk([1:ka],[ma+me+la+1:ma+me+la+le]);
deak=dk([ka+1:ka+ke],[1:ma]);deyak=dk([ka+1:ka+ke],[ma+me+1:ma+me+la]);
dek=dk([ka+1:ka+ke],[ma+1:ma+me]);
dyak=dk([1:ka],[ma+me+1:ma+me+la]);
dyek=dk([ka+1:ka+ke],[ma+la+me+1]:[ma+la+me+le]);

// HERE Construct the state space representation of Tcent from input: zac to
// output:  zea using the system with the global controller.  This
// is used in evaluating the zea tracking cost, but does not alter
// with the choice of parameters so it is constructed once only
// in this routine and then stored.
ag= [ak,   (-bak*cap-bek*cep+byak*cyap+byek*cyep);...
       (bpa*cak+ bpe*cek),  (ap-(bpa*dak*cap)-(bpe*dek*cep)+...
      bpa*dyak*cyap+bpe*dyek*cyep-bpa*daek*cep+bpa*dayek*cyep...
      -bpe*deak*cap+bpe*deyak*cyap)];

bg= [bak;  bpa*dak];
cg= [0*ones(pea,nk)  ceap];
dg= 0*ones(pea,ma);

// Let sg be the constructed state space representation of Tcent
sg=[ag,bg;cg,dg];
ng= nk+np;

// If Ds are fixed, their initial values are determined here
if aore<>1,if (fixd-1)*(fixd-2)=0,da=[dak,dyak;da([ka+1:ka+pea],:)];end;end;

if aore <> 2, if (fixd-2)*(fixd-3)=0, de(:,[1:pea])=0*ones(ke,pea); end;...
  if (fixd-1)*(fixd-2)=0,  de=[de(:,[1:pea]),dek,dyek]; end;...
end;

daa=da([1:ka],[1:ma]); daya=da([1:ka],[ma+1:ma+la]);
dee=de(:,[pea+1:pea+me]); deye=de(:,[pea+me+1:metot]);
ska=[aa, ba; ca, da]; ske=[ae,be; ce, de];


// Store the current (initial) values of ska and ske in ska0 and ske0
// to be used if the subcontroller(s) are fixed.
ska0=ska;ske0=ske;

// The stability upperbound stabil for the A submatrices should
// be read in from init.dat and later stored in const.dat
// If it is not defined or it is defined too high, it is given a
// default value
if exist('stabil')=0 then stabil=-1e-9;
if stabil>-1e-9, stabil=-1e-9;

// Save the constants in const.dat
save 'const.dat' la le lya lye ka ma ke me pea nka nke sp sk np nk frq ...
   omega daa dee daya deye atot sg ng swi nwi swo nwo weight stabil ...
   bal bel ska0 ske0 aore Nperf Ntrack fixd;

// if there was no error in the data checking
p_i=longcol(ska,ske);
[row,col]=size(p_i);
hi=eye(row);
fx=0*ones(atot,1);

for I=1:atot, ....
  if p_i(I)>stabil, ...
    p_i(I)=stabil; ...
    fx(I)=1; ...
    hi(I,I)=0;...
end;end;

// Initialize costs and gradients
[gradi,grad0,grad1,fh,jh0,jh1]=cost(p_i,lambda);

// Save intermediate results
save 'inter.dat' p_i gradi grad0 grad1 fh jh0 jh1 hi lambda fx;

// Save starting data for restarts
save 'start.dat' aore p_i gradi grad0 grad1 fh jh0 jh1 lambda fx;

// call the PARTITION routine returning the final optimized vector
p_opt=PARTITION(stop);

// Generate the optimized ska and ske as the final output of the program
[ska_opt,ske_opt]=MAT(p_opt);
```

```
load 'inter.dat' p_i fh jh0 jh1 grad1 grad0 grad1 hi fx lambda;
// If a dummy feedback for ya or ye was added, then remove it
if lya=0, [r,c]=size(ska_opt); ska_opt=ska_opt(:,1:c-1); end;
if lye=0, [r,c]=size(ske_opt); ske_opt=ske_opt(:,1:c-1); end;
// save final data
save 'inter.dat' p_i fh jh0 jh1 grad1 grad0 grad1 hi fx lambda ska_opt ske_opt;

retf

// The UDF start.mtx is the main routine for the controller partitioning
// program. It expects the input to the program to be in the datafile
// 'init.dat', and requires certain data to be there.
// 'init.dat' contains the global controller sc, the initial partitioned
// controllers s_ka and s_ke, and the plant sp along with their orders nsc,
// ns_ka, ns_ke, and np resp.  It also contains the number of intermediate
// variables pea and the range of frequencies for calculating norms frq. The
// optional weighting matrices swi and swo (for input and/or output weighting)
// may be included (default = Id in both cases) as may la and/or le, the
// numbers of neg. feedback variables (if these are absent, then the code
// inserts dummy values of 1 and loads appropriate zeros in the data matrices.)
// stabil = upper bound for eigenvalues of Aa and Ae to assure stability is
// also contained in init.dat (if missing, defaults to -1e-9).
//
// The code gives the option of fixing values of either subcontroller if
// optimization over the parameters in only one subcontroller is desired.
//
// The option is also given to fix certain of the D submatrices.
//
// The subcontrollers are put into a modified modal form where the A matrices
// consist of 2x2 companion blocks and the first column of the B matrix is
// fixed to its initial value (or 1e-9 if this initial value is zero).  These
// are then split to identify the state space representations from inputs to
// outputs using intermediate variables.
// Here the modifications are made to account for the absence of la and/or le.
//
// The logarithmic frequency range over which the cost function is evaluated is
// created as Omega.  The vector of Weights used in Simpson's Rule integration
// is also calculated.
//
// The state-space representation for the  nominal "command-tracking"
// transfer matrix Tcent:zac to zea is computed, sg.  Its row norms are stored
// to be used as normalizations in the fTrack part of the cost function.
//
// The file 'const.dat' contains constants --- the numbers of subcontroller
// inputs, ma and me; outputs, ka and ke; and intermediate variables, pea;
// the numbers of negative feedbacks from the plant (lya/lye relects the actual
// numbers, i.e. 0 if la resp le were absent in 'init.dat'.  In this case la/le
// would contain the dummy value 1).  Further constants are the state space
// matrices for the plant, sp; the centralized controller, sk; the
// nominal tracking transfer matrix, sg; the frequency weighting matrices,
// swi and swo; and their orders --- np, nk, ng, and nw along with the orders
// of the subcontrollers nka and nke; plus the frequency range, frq; the
// vector of frequencies, omega; the weights for Simpson's Rule, weight; and
// the vector of row norms for the nominal tracking transfer matrix.  Other
// constants are the values of the D subcontroller submatrices which are fixed
// equal to the corresponding submatrices of the state space representation
// for the centralized controller; the first columns of the subcontroller B
// matrices which are fixed by the modl canonical form; and finally, the
// values of ska0 and ske0 which are used in case the appropriate subcontroller
// is fixed.  atot indicates the sum of orders of the subcontroller state-space
// matrices which are not fixed.
//
// The initial system matrices for the subcontrollers ska0 and ske0 are put
// into the form of a parameter vector p_i, an identity matrix is created as
// the initial approximation for the inverse Hessian used in the optimization
// process and a vector, fx, indicating which of the entries in the state-space
// A matrices are at the stability upper bound.  p_i and lambda are passed to
// the COST function which returns the initial values of:
//    fh = the total cost
//    jho = the performance cost
//    jh1 = the tracking cost
//    grad1 = gradient of the total cost
//    grad0 = gradient of the performance cost
//    grad1 = gradient of the tracking cost
//
// These values are stored in 'inter.dat' the file for intermediate results.
// 'start.dat' contains a record of the starting configuration.
//
// The parameters are passed to PARTITION which carries out the optimization.
// This procedure reads the data it needs from the file 'inter.dat' and
// returns the optimized paramter vector p_opt.
// This final vector is split into the optimized airframe and engine
// controllers ska_opt and ske_opt, which are the final output of
// the program.

// Created by Phil Schmidt and students Nader Kamrani and Brian Holawecky at
// The University of Akron with the support of NASA Lewis Research Center
// under grant NAG-3-1146.
```

```
//<zer> = z(nrow,ncol)
// Construct a matrix of zeros of size nrow x ncol.  This is done so that
// less space is taken within the main program
// Input:
//   nrow - number of rows
//   ncol - number of columns
// Output:
//   zer - the zero matrix of the desired size (nrow,ncol)

zer=0*ones(nrow,ncol);

retf

// Created by Phil Schmidt at The University of Akron with the support
// of NASA Lewis Research Center under grant NAG-3-1146.
```

Appendix IV

Data for Example

The data file INIT.DAT corresponding to the example contains the following:

PEA = 1 — number of interface variables

FRQ = [0.1; 100; 41] — describes the frequency range of 41 points between 0.1 and 100

NP = 13 — order of the integrated plant

NSC = 13 — order of the centralized controller

NS_KA = 10 — order of the airframe subcontroller

NS_KE = 7 — order of the engine subcontroller

NWI = 26 — order of the input weighting transfer matrix

NTRACK =

$$\begin{bmatrix} 6.1823\text{D}-01 & 6.3153\text{D}-01 & 6.4195\text{D}-01 & 6.5131\text{D}-01 & 6.6139\text{D}-01 & 6.7411\text{D}-01 \cdots \\ \cdots & 6.9179\text{D}-01 & 7.1732\text{D}-01 & 7.5413\text{D}-01 & 8.0582\text{D}-01 & 8.7492\text{D}-01 \cdots \\ \cdots & 9.6037\text{D}-01 & 1.0534\text{D}+00 & 1.1326\text{D}+00 & 1.1646\text{D}+00 & 1.1194\text{D}+00 \cdots \\ \cdots & 9.9744\text{D}-01 & 8.3547\text{D}-01 & 6.8055\text{D}-01 & 5.6432\text{D}-01 & 4.9513\text{D}-01 \cdots \\ \cdots & 4.6326\text{D}-01 & 4.5302\text{D}-01 & 4.5255\text{D}-01 & 4.5595\text{D}-01 & 4.6188\text{D}-01 \cdots \\ \cdots & 4.7282\text{D}-01 & 4.9653\text{D}-01 & 5.5002\text{D}-01 & 6.6494\text{D}-01 & 8.9018\text{D}-01 \cdots \\ \cdots & 1.2933\text{D}+00 & 1.9711\text{D}+00 & 3.0733\text{D}+00 & 4.8371\text{D}+00 & 7.6375\text{D}+00 \cdots \\ \cdots & 1.2067\text{D}+01 & 1.9074\text{D}+01 & 3.0171\text{D}+01 & 4.7762\text{D}+01 & 7.5645\text{D}+01 \end{bmatrix}^{\mathrm{T}}$$

— 41 × 1 vector of tracking weights (The size of NTRACK is FRQ(3)× PEA.)

SP (listed below) — state-space matrix for the integrated plant transfer matrix

SC (listed below) — state-space matrix for the centralized controller transfer matrix

S_KA (listed below) — state-space matrix for the initial airframe subcontroller transfer matrix

S_KE (listed below) — state-space matrix for the initial engine subcontroller transfer matrix

SWI (described below) — state-space matrix for the input weighting transfer matrix transfer matrix

The data matrices and initial partitioning matrices for the controller partitioning example are listed below. In all cases the given matrices A, B, C and D correspond to the state-space representation of the given system or subsystem

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = A\mathbf{x} + B\mathbf{y}$$
$$\mathbf{u} = C\mathbf{x} + D\mathbf{y}.$$

The state-space matrix for this system is S where $S = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$.

The integrated airframe propulsion system with integrator augmentation is represented by $SP = \begin{bmatrix} AP & BP \\ CP & DP \end{bmatrix}$ where

$$AP = \begin{bmatrix}
-4.40E^{-2} & 3.60E^{-2} & -3.85E^{+1} & -3.18E^{+1} & 1.40E^{-2} & 3.14E^{-4} & 2.59E^{-4} & 3.81E^{-2} & \\
-2.27E^{-1} & -4.46E^{-1} & 1.94E^{+2} & -4.59E^{+0} & 5.19E^{-4} & -1.57E^{-5} & -2.10E^{-6} & 1.82E^{-4} & \\
-3.09E^{-3} & 1.51E^{-2} & -1.94E^{-1} & -4.81E^{-4} & 2.56E^{-5} & 9.46E^{-7} & 3.74E^{-7} & 3.66E^{-5} & \\
0 & 0 & 1.00E^{+0} & 0 & 0 & 0 & 0 & 0 & \\
1.42E^{-1} & -9.89E^{-1} & 0 & 2.00E^{+2} & 0 & 0 & 0 & 0 & \\
7.78E^{-1} & 1.54E^{-1} & 0 & 0 & -8.48E^{-2} & -4.19E^{+0} & 6.02E^{+0} & -3.43E^{+2} & \\
1.51E^{-1} & 3.00E^{-2} & 0 & 0 & -1.65E^{-2} & 4.26E^{-1} & -5.70E^{+0} & 2.71E^{+1} & \cdots \\
7.93E^{-1} & 1.57E^{-1} & 0 & 0 & -3.50E^{-1} & 2.29E^{-1} & 1.15E^{-1} & -9.02E^{+1} & \\
-1.00E^{-1} & -1.99E^{-2} & 0 & 0 & 1.09E^{-2} & 3.74E^{-2} & -1.03E^{-1} & -7.95E^{+0} & \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 &
\end{bmatrix}$$

$$\begin{bmatrix}
2.25E^{-3} & 1.71E^{-1} & -1.02E^{+1} & 6.91E^{+0} & -4.18E^{-3} \\
-2.95E^{-6} & 6.17E^{-5} & -1.46E^{-2} & 7.10E^{-3} & -5.45E^{+0} \\
2.67E^{-6} & 2.75E^{-4} & 5.33E^{-3} & -8.38E^{-3} & -7.97E^{-1} \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
1.16E^{+1} & 7.34E^{+2} & 0 & 0 & 0 \\
\cdots \; 1.03E^{+1} & 2.68E^{+2} & 0 & 0 & 0 \\
8.47E^{-1} & 9.06E^{+1} & -2.15E^{+3} & -2.58E^{+3} & 0 \\
-1.06E^{+0} & 8.21E^{+2} & 0 & 0 & 0 \\
0 & -1.00E^{-2} & 0 & 0 & 0 \\
0 & 0 & -1.00E^{-2} & 0 & 0 \\
0 & 0 & 0 & -1.00E^{-2} & 0 \\
0 & 0 & 0 & 0 & -1.00E^{-2}
\end{bmatrix}$$

$$BP = \begin{bmatrix}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 1.00E^{+0} & 0 & 0 \\
0 & 0 & 1.00E^{+0} & 0 \\
0 & 0 & 0 & 1.00E^{+0} \\
1.00E^{+0} & 0 & 0 & 0
\end{bmatrix} \qquad DP = \begin{bmatrix}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}$$

$$CP = \begin{bmatrix}
4.89E^{-2} & 9.70E^{-3} & 0 & 0 & 0 & 0 & 0 & 0 & \\
0 & 0 & 1.71E^{+1} & 1.71E^{+0} & 0 & 0 & 0 & 0 & \\
0 & 0 & 0 & 0 & 0 & 1.74E^{-3} & 0 & 0 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 2.04E^{-1} & \\
1.21E^{-4} & 2.40E^{-5} & 0 & 0 & 1.20E^{-3} & 3.45E^{-5} & 2.31E^{-5} & 3.15E^{-3} &
\end{bmatrix}$$

$$\begin{bmatrix}
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
\cdots \; 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
1.92E^{-4} & 1.47E^{-3} & -8.53E^{-1} & 5.68E^{-1} & 0
\end{bmatrix}$$

The augmented plant state variables are

$$\mathbf{x} = [u, w, q, \theta, h, N2R, N25, P6, T41B, \delta_{\mathrm{TV}}, WF, A78, A8]$$

where

u = aircraft body axis forward velocity, ft/s

w = aircraft body axis vertical velocity, ft/s

q = aircraft pitch rate, rad/s

θ = pitch attitude, deg

h = altitude, ft

$N2R$ = engine fan speed, rpm

$N25$ = core compressor speed, rpm

$P6$ = engine mixing plane pressure, psia

$T41B$ = engine high pressure turbine blade temperature, ° R

δ_{TV} = normalized thrust vectoring angle, deg/10

WF = normalized engine main burner fuel flow rate, lb m/hr/500

$A78$ = normalized thrust reverser port area, $\mathrm{in}^2/50$

$A8$ = normalized main nozzle throat area, $\mathrm{in}^2/100$.

The augmented plant outputs are listed in the order $\begin{bmatrix} \mathbf{z}_a \\ \mathbf{z}_e \\ \mathbf{z}_{ea} \end{bmatrix}$ with $\mathbf{z}_a = \begin{bmatrix} V \\ q_v \end{bmatrix}$, $\mathbf{z}_e = \begin{bmatrix} N2 \\ EPR \end{bmatrix}$, and $\mathbf{z}_{ea} = FEX$ where

V = normalized aircraft speed, ft/s/20

q_v = normalized pitch variable, $(q(\mathrm{deg/s})+0.1\theta(\mathrm{deg}))/3$

$N2$ = normalized engine fan speed, % of maximum allowable rpm at operating condition/5

EPR = normalized engine pressure ratio, ratio/0.3

The augmented plant inputs are the rates of change on normalized control variables as described in the text.

The centralized controller has state-space representation $SC = \begin{bmatrix} AC & BC \\ CC & DC \end{bmatrix}$ where

$$AC = \begin{bmatrix}
-1.23E^{+0} & -2.00E^{-1} & 3.63E^{+1} & -2.43E^{+1} & 1.40E^{-2} & -4.41E^{-4} & 2.59E^{-4} & -5.38E^{-2} & \\
-3.21E^{-2} & -4.07E^{-1} & -5.53E^{+1} & -2.96E^{+1} & 5.19E^{-4} & 1.59E^{-5} & -2.10E^{-6} & 1.21E^{-3} & \\
-2.89E^{-3} & 1.52E^{-2} & -6.49E^{+0} & -6.30E^{-1} & 2.56E^{-5} & 1.08E^{-6} & 3.74E^{-7} & -8.20E^{-6} & \\
3.75E^{-5} & 7.43E^{-6} & -1.34E^{-2} & -1.01E^{-1} & 0 & -1.80E^{-7} & 0 & 4.50E^{-5} & \\
-4.36E^{-1} & -1.10E^{+0} & 2.02E^{+1} & 2.02E^{+2} & 0 & 1.05E^{-4} & 0 & 3.17E^{-3} & \\
1.86E^{-1} & 3.70E^{-2} & 1.14E^{+1} & 1.14E^{+0} & -8.48E^{-2} & -7.78E^{+0} & 6.02E^{+0} & -2.20E^{+2} & \\
-1.78E^{-1} & -3.53E^{-2} & 1.07E^{+0} & 1.07E^{-1} & -1.65E^{-2} & -9.61E^{-1} & -5.70E^{+0} & -2.47E^{+1} & \cdots \\
7.88E^{-1} & 1.56E^{-1} & -2.86E^{-1} & -2.86E^{-2} & -3.50E^{-1} & 2.38E^{-1} & 1.15E^{-1} & -9.87E^{+1} & \\
-2.46E^{-1} & -4.88E^{-2} & -1.26E^{+0} & -1.26E^{-1} & 1.09E^{-2} & -7.49E^{-1} & -1.03E^{-1} & -2.86E^{+1} & \\
-1.32E^{-1} & 1.00E^{+0} & 1.73E^{+3} & 1.73E^{+2} & 1.79E^{-3} & 2.70E^{-5} & 8.48E^{-6} & -8.94E^{-4} & \\
-3.77E^{-2} & -7.60E^{-3} & -4.04E^{-1} & 7.47E^{-2} & 1.76E^{-3} & -1.09E^{-1} & -6.16E^{-2} & -2.72E^{-1} & \\
3.80E^{+0} & 7.08E^{-1} & -2.26E^{+1} & -2.41E^{+1} & -9.57E^{-3} & -1.18E^{-2} & -1.24E^{-3} & 8.23E^{+0} & \\
-3.06E^{+0} & -5.68E^{-1} & 1.96E^{+1} & 2.00E^{+1} & -3.12E^{-2} & -1.39E^{-2} & -1.29E^{-3} & 9.88E^{+0} &
\end{bmatrix}$$

$$\begin{bmatrix}
2.25E^{-3} & -4.18E^{-3} & 1.71E^{-1} & -1.02E^{+1} & 6.91E^{+0} \\
-2.95E^{-6} & -5.45E^{+0} & 6.17E^{-5} & -1.46E^{-2} & 7.10E^{-3} \\
2.67E^{-6} & -7.97E^{-1} & 2.75E^{-4} & 5.33E^{-3} & -8.38E^{-3} \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
1.16E^{+1} & 0 & 7.34E^{+2} & 0 & 0 \\
\cdots \quad 1.03E^{+1} & 0 & 2.68E^{+2} & 0 & 0 \\
8.47E^{-1} & 0 & 9.06E^{+1} & -2.15E^{+3} & -2.58E^{+3} \\
-1.06E^{+0} & 0 & 8.21E^{+2} & 0 & 0 \\
9.44E^{-5} & -5.24E^{+1} & 1.06E^{-2} & 4.16E^{-1} & -3.58E^{-1} \\
-2.14E^{-1} & 1.06E^{-2} & -2.33E^{+1} & 3.64E^{+0} & 4.31E^{+0} \\
2.76E^{-2} & 4.16E^{-1} & 3.64E^{+0} & -1.25E^{+2} & -1.37E^{+2} \\
3.27E^{-2} & -3.58E^{-1} & 4.31E^{+0} & -1.37E^{+2} & -1.76E^{+2}
\end{bmatrix}$$

$$BC = \begin{bmatrix}
2.43E^{+1} & -4.36E^{+0} & 4.33E^{-1} & 4.51E^{-1} \\
-3.99E^{+0} & 1.45E^{+1} & -1.81E^{-2} & -5.07E^{-3} \\
-4.13E^{-3} & 3.66E^{-1} & -7.81E^{-5} & 2.19E^{-4} \\
-7.66E^{-4} & 5.89E^{-2} & 1.03E^{-4} & -2.20E^{-4} \\
1.18E^{+1} & -1.17E^{+0} & -6.02E^{-2} & -1.55E^{-2} \\
1.20E^{+1} & -6.68E^{-1} & 2.05E^{+3} & -6.00E^{+2} \\
6.74E^{+0} & -6.23E^{-2} & 7.95E^{+2} & 2.54E^{+2} \\
1.08E^{-1} & 1.66E^{-2} & -5.12E^{+0} & 4.16E^{+1} \\
2.97E^{+0} & 7.34E^{-2} & 4.50E^{+2} & 1.01E^{+2} \\
-1.55E^{-1} & -1.24E^{+0} & 1.74E^{-3} & 9.36E^{-4} \\
1.05E^{-2} & 6.63E^{-4} & 1.16E^{+0} & 4.47E^{-1} \\
-9.50E^{-1} & 1.18E^{-1} & 2.94E^{-1} & -7.46E^{-1} \\
7.96E^{-1} & -9.98E^{-2} & 3.36E^{-1} & -8.97E^{-1}
\end{bmatrix} \qquad DC = \begin{bmatrix}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}$$

$$CC = \begin{bmatrix}
1.40E^{-1} & -1.00E^{+0} & -1.71E^{+3} & -1.71E^{+2} & -1.79E^{-3} & -3.01E^{-5} & -8.48E^{-6} & 7.03E^{-4} & \\
3.72E^{-2} & 7.50E^{-3} & 3.93E^{-1} & -7.58E^{-2} & -1.76E^{-3} & 1.07E^{-1} & 6.16E^{-2} & 1.81E^{-1} & \cdots \\
-3.76E^{+0} & -6.99E^{-1} & 2.05E^{+1} & 2.39E^{+1} & 9.57E^{-3} & 1.12E^{-2} & 1.24E^{-3} & -8.08E^{+0} & \\
3.02E^{+0} & 5.60E^{-1} & -1.79E^{+1} & -1.98E^{+1} & 3.12E^{-2} & 1.33E^{-2} & 1.29E^{-3} & -9.70E^{+0} &
\end{bmatrix}$$

$$\begin{bmatrix}
-9.44E^{-5} & 5.24E^{+1} & -1.06E^{-2} & -4.16E^{-1} & 3.58E^{-1} \\
\cdots \quad 2.14E^{-1} & -1.06E^{-2} & 2.33E^{+1} & -3.64E^{+0} & -4.31E^{+0} \\
-2.76E^{-2} & -4.16E^{-1} & -3.64E^{+0} & 1.25E^{+2} & 1.37E^{+2} \\
-3.27E^{-2} & 3.58E^{-1} & -4.31E^{+0} & 1.37E^{+2} & 1.76E^{+2}
\end{bmatrix}$$

The initial partitioning consisted of the airframe controller with representation S_KA = $\begin{bmatrix} \text{A\_KA} & \text{B\_KA} \\ \text{C\_KA} & \text{D\_KA} \end{bmatrix}$ of order 10 and an engine controller with representation S_KE = $\begin{bmatrix} \text{A\_KE} & \text{B\_KE} \\ \text{C\_KE} & \text{D\_KE} \end{bmatrix}$ of order 7. The initial partitioning was obtained by the procedure of reference [**2**]. This partitioning is not in the minimal parameter form. The submatrices in the initial partitioning are:

$$\text{A\_KA} = \begin{bmatrix} -5.87E^{+1} & -4.14E^{+1} & 1.71E^{+0} & 0 & 0 & 0 & 0 & \cdots \\ 4.14E^{+1} & -7.77E^{-2} & 6.28E^{-2} & 0 & 0 & 0 & 0 & \\ -1.63E^{+0} & 5.71E^{-2} & -5.76E^{-1} & 0 & 0 & 0 & 0 & \\ 0 & 0 & 0 & -1.21E^{+1} & -8.59E^{+0} & 3.37E^{-1} & 1.36E^{-1} & \\ 0 & 0 & 0 & 8.55E^{+0} & -1.75E^{-2} & 1.38E^{-2} & 5.33E^{-3} & \\ 0 & 0 & 0 & -2.16E^{-2} & -5.98E^{-3} & -3.88E^{-2} & -1.95E^{-1} & \\ 0 & 0 & 0 & -1.22E^{-2} & 1.96E^{-3} & 1.95E^{-1} & -7.55E^{-3} & \\ 0 & 0 & 0 & 6.95E^{-2} & -9.91E^{-3} & -1.98E^{-1} & 4.26E^{-2} & \\ 0 & 0 & 0 & -7.86E^{-2} & 4.73E^{-3} & 2.83E^{-2} & -1.34E^{-2} & \\ 0 & 0 & 0 & 1.14E^{+3} & 4.13E^{+1} & -1.58E^{+1} & -6.42E^{+0} & \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ -5.52E^{-1} & 1.44E^{-1} & 0 \\ -2.18E^{-2} & 5.76E^{-3} & 0 \\ 1.98E^{-1} & -3.37E^{-2} & 0 \\ 4.28E^{-2} & -1.57E^{-2} & 0 \\ -2.87E^{-1} & 1.45E^{-1} & 0 \\ 1.28E^{-1} & -4.61E^{-1} & 0 \\ 2.60E^{+1} & -6.80E^{+0} & -1.00E^{-2} \end{bmatrix}$$

$$\text{B\_KA} = \begin{bmatrix} 2.90E^{-1} & -2.68E^{+1} \\ 9.67E^{-2} & 9.22E^{-1} \\ 1.19E^{-1} & -3.63E^{-1} \\ 1.13E^{+3} & -1.04E^{+2} \\ -3.66E^{+1} & 1.90E^{+1} \\ 1.51E^{+0} & 1.58E^{+1} \\ -3.36E^{-1} & -6.41E^{+0} \\ -3.35E^{-1} & 2.59E^{+1} \\ 3.16E^{+0} & -6.02E^{+0} \\ 0 & 0 \end{bmatrix} \qquad \text{D\_KA} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$\text{C\_KA} = \begin{bmatrix} 2.68E^{+1} & 9.27E^{-1} & -3.83E^{-1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1.00E^{-4} \end{bmatrix}$$

and

$$\text{A\_KE} = \begin{bmatrix} -3.50E^{+2} & 5.11E^{+1} & 0 & 0 & 0 & 0 & 0 \\ -5.30E^{+1} & -1.75E^{-2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1.67E^{-1} & -1.90E^{+1} & -1.04E^{+0} & -4.51E^{-1} & 2.83E^{-1} \\ 0 & 0 & 1.73E^{+1} & -2.02E^{+1} & -1.06E^{+1} & -1.03E^{+0} & -6.57E^{-2} \\ 0 & 0 & 1.17E^{+0} & -1.06E^{+1} & -1.92E^{+1} & 1.89E^{+1} & -2.99E^{+1} \\ 0 & 0 & -5.29E^{+0} & 1.41E^{+1} & 1.03E^{+2} & -3.59E^{+2} & 2.50E^{+2} \\ 0 & 0 & -8.39E^{-1} & 5.67E^{+0} & 3.56E^{+1} & -2.46E^{+2} & -2.07E^{+1} \end{bmatrix}$$

$$\text{B\_KE} = \begin{bmatrix} -8.66E^{+1} & 0 & 0 \\ -5.94E^{-1} & 0 & 0 \\ 0 & 1.69E^{-2} & -1.83E^{+0} \\ 0 & -1.91E^{+1} & 7.38E^{-1} \\ 0 & -5.75E^{+0} & 5.45E^{+0} \\ 0 & 5.49E^{+0} & -2.93E^{+1} \\ 0 & 2.51E^{+0} & -4.48E^{+0} \end{bmatrix} \qquad \text{D\_KE} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\text{C\_KE} = \begin{bmatrix} -4.12E^{-1} & -1.44E^{-1} & -1.81E^{+0} & -1.91E^{+1} & -5.61E^{+0} & -1.45E^{+0} & 4.77E^{-1} \\ 7.47E^{+1} & 1.45E^{-1} & 1.86E^{-1} & -6.92E^{-1} & -3.67E^{+0} & 1.91E^{+1} & -3.44E^{+0} \\ -4.37E^{+1} & 5.58E^{-1} & 2.15E^{-1} & -9.85E^{-1} & -4.22E^{+0} & 2.28E^{+1} & -3.79E^{+0} \end{bmatrix}$$

Recall that the input weighting transfer matrix is $W_i(s) = G(s)(I + K(s)G(s))^{-1}$ where $G(s)$ and $K(s)$ are the integrated plant and centralized controller transfer matrices respectively. The state-space representation for this transfer matrix is too large to list. Instead, the user can easily construct it from the following MATRIX$_X$ command applied to the state space matrices for the integrated plant and the centralized controller

[SWI,NWI]=FEEDBACK(SC,NSC,SP,NP).

Notice that this command also produces the correct value for NWI, the order of the weighting transfer matrix.

The parameter optimization algorithm for controller partitioning was applied to the problem with the initial partitioning given by S_KA and S_KE as listed above and with input weighting SWI as described above. The controllers obtained from this process had state-space representations SKA_OPT for the airframe and SKE_OPT for the engine where, as before

$$\text{A\_A\_OPT} = \begin{bmatrix} 0 & 1.00E^{+0} & 0 & 0 & 0 & 0 & 0 & & \\ -4.43E^{-2} & -3.15E^{-1} & 0 & 0 & 0 & 0 & 0 & & \\ 0 & 0 & 0 & 1.00E^{+0} & 0 & 0 & 0 & & \\ 0 & 0 & -7.28E^{+1} & -1.12E^{+1} & 0 & 0 & 0 & & \\ 0 & 0 & 0 & 0 & 0 & 1.00E^{+0} & 0 & \cdots & \\ 0 & 0 & 0 & 0 & -1.72E^{+3} & -5.91E^{+1} & 0 & & \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & & \\ 0 & 0 & 0 & 0 & 0 & 0 & -1.00E^{-5} & & \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & & \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & & \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \cdots \quad 0 & 0 & 0 \\ 1.00E^{+0} & 0 & 0 \\ -3.41E^{+0} & 0 & 0 \\ 0 & 0 & 1.00E^{+0} \\ 0 & -9.96E^{-3} & -1.99E^{-1} \end{bmatrix}$$

$$\text{B\_A\_OPT} = \begin{bmatrix} -3.67E^{-3} & -5.45E^{+3} \\ -1.24E^{-2} & 1.60E^{+3} \\ 1.31E^{+5} & -9.08E^{+3} \\ 8.50E^{+4} & -2.58E^{+4} \\ -4.23E^{-1} & 2.03E^{+1} \\ -6.84E^{+0} & 9.54E^{+2} \\ 1.03E^{+4} & -1.84E^{+3} \\ -2.11E^{+2} & 4.55E^{+2} \\ 2.14E^{-1} & 6.51E^{+3} \\ -7.97E^{-2} & -6.50E^{+2} \end{bmatrix} \qquad \text{D\_A\_OPT} = \begin{bmatrix} 2.82E^{-1} & -2.94E^{-3} \\ 8.98E^{-4} & -1.59E^{-3} \end{bmatrix}$$

$$\text{C\_A\_OPT} = \begin{bmatrix} -3.86E^{-5} & -1.06E^{-4} & -4.69E^{-5} & -5.57E^{-6} & -9.82E^{+0} & -5.41E^{-1} & -1.85E^{-6} & \cdots \\ 2.49E^{-4} & 2.03E^{-3} & 1.91E^{-6} & -1.28E^{-5} & -2.99E^{-3} & 2.17E^{-4} & 7.65E^{-5} & \cdots \end{bmatrix}$$

$$\begin{matrix} \cdots & 4.09E^{-4} & 6.44E^{-2} & 6.46E^{-1} \\ \cdots & -2.63E^{-4} & -1.59E^{-2} & -1.57E^{-1} \end{matrix}$$

The optimized engine controller has state-space representation

$$\text{A\_E\_OPT} = \begin{bmatrix} 0 & 1.00E^{+0} & 0 & 0 & 0 & 0 & 0 \\ -2.12E^{+2} & -1.65E^{+1} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.00E^{+0} & 0 & 0 & 0 \\ 0 & 0 & -7.15E^{+4} & -3.89E^{+2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.00E^{+0} & 0 \\ 0 & 0 & 0 & 0 & -1.20E^{+2} & -2.35E^{+1} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -3.41E^{+2} \end{bmatrix}$$

$$\text{B\_E\_OPT} = \begin{bmatrix} 1.00E^{-9} & -2.53E^{+1} & 5.04E^{+0} \\ 1.00E^{-9} & 9.38E^{+1} & -7.78E^{+1} \\ 1.00E^{-9} & 3.75E^{+0} & -1.56E^{+1} \\ 1.00E^{-9} & 1.67E^{+3} & -7.69E^{+3} \\ 1.32E^{+1} & -1.35E^{+1} & 2.95E^{+0} \\ -1.05E^{+2} & 1.94E^{+2} & -4.60E^{+1} \\ 8.96E^{+1} & -3.56E^{-1} & 8.92E^{-2} \end{bmatrix} \qquad \text{D\_E\_OPT} = \begin{bmatrix} 5.82E^{-1} & 1.72E^{+0} & 1.41E^{-2} \\ -2.04E^{+0} & 1.48E^{+0} & 1.10E^{-1} \\ 6.64E^{+0} & -6.93E^{-1} & 1.03E^{-2} \end{bmatrix}$$

$$\text{C\_E\_OPT} = \begin{bmatrix} -4.65E^{+0} & -1.07E^{+0} & -1.42E^{-1} & -9.50E^{-4} & 2.52E^{+1} & 3.36E^{+0} & 5.68E^{-1} \\ 1.97E^{+0} & 1.44E^{-1} & 8.80E^{+0} & 5.88E^{-2} & 2.14E^{+1} & 1.46E^{+0} & -7.44E^{+1} \\ -1.62E^{+0} & -1.10E^{-1} & 1.01E^{+1} & 6.27E^{-2} & -1.92E^{+1} & -1.37E^{+0} & 4.47E^{+1} \end{bmatrix}$$

Residualization of high frequency modes was applied to the optimized engine controller to reduce it to one with order 4 (not shown here because it is easily obtained). Balanced model reduction was applied to the optimized airframe controller to reduce it to one of order 6. The optimization procedure was applied to this sixth order subcontroller with the engine controller fixed at the one of fourth order. The resulting reduced order optimized airframe subcontroller is

$$SK_{Ared} = \left[\begin{array}{cccccc|cc} 0 & 1.00E^{+0} & 0 & 0 & 0 & 0 & -2.81E^{-1} & 8.98E^{-2} \\ -2.84E^{+0} & -4.39E^{+1} & 0 & 0 & 0 & 0 & -1.39E^{-2} & 1.18E^{-1} \\ 0 & 0 & 0 & 1.00E^{+0} & 0 & 0 & 1.29E^{+0} & -5.50E^{+0} \\ 0 & 0 & -1.71E^{+3} & -6.19E^{+1} & 0 & 0 & -1.53E^{+1} & -9.18E^{+2} \\ 0 & 0 & 0 & 0 & 0 & 1.00E^{+0} & 1.38E^{+0} & -1.20E^{-1} \\ 0 & 0 & 0 & 0 & -6.92E^{+1} & -1.18E^{+1} & -5.03E^{+0} & 5.95E^{-1} \\ \hline -1.87E^{-1} & -7.05E^{+0} & 7.83E^{+0} & 7.95E^{-1} & 3.43E^{+1} & 1.28E^{+0} & -3.24E^{+0} & 5.62E^{-1} \\ -1.95E^{+0} & 1.40E^{+1} & 2.99E^{-2} & 8.66E^{-4} & -8.63E^{+0} & -2.28E^{+0} & 1.60E^{-3} & -2.25E^{-3} \end{array}\right]$$

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1994	Final Contractor Report

4. TITLE AND SUBTITLE
Computer Code for Controller Partitioning With IFPC Application
A User's Manual

5. FUNDING NUMBERS
WU–505–62–50
C–NAG3–1146 Errata

6. AUTHOR(S)
Phillip H. Schmidt and Asim Yarkhan

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of Akron
Akron, Ohio 44325

8. PERFORMING ORGANIZATION REPORT NUMBER
E–8654

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Lewis Research Center
Cleveland, Ohio 44135–3191

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
NASA CR–195291

11. SUPPLEMENTARY NOTES
Project Manager, Sanjay Garg, Instrumentation and Control Technology Division, organization code 2550, NASA Lewis Research Center, (216) 433–2355.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified - Unlimited
Subject Categories 08 and 63

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 words)***

This is a user's manual for the computer code for partitioning a centralized controller into decentralized subcontrollers with applicability to Integrated Flight/Propulsion Control (IFPC). Partitioning of a centralized controller into two subcontrollers is described and the algorithm on which the code is based is discussed. The algorithm uses parameter optimization of a cost function which is described here. The major data structures and functions are described. Specific instructions are given. The user is led through an example of an IFPC application.

14. SUBJECT TERMS
Integrated control; Centralized control; Decentralized control; Flight control; Propulsion control

15. NUMBER OF PAGES
76

16. PRICE CODE
A04

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	